MODULE 3

1. First-order logic

- Representation revisited
- Syntax and semantics of first-order logic
- Using first-order logic
- Knowledge engineering in first-order logic

2. Inference in first-order logic

- Propositional versus first-order inference
- Uniform and lifting
- Forward chaining
- Backward chaining
- Resolution

FIRST-ORDER LOGIC

Representation revisited

Combining the best of formal and natural languages

The syntax of natural language

- nouns and noun phrases that refer to **objects** (squares, pits, wumpuses)
- verbs and verb phrases that refer to **relations** among objects (is breezy, is adjacent to, shoots).
- Some of these relations are **functions**—relations in which there is only one "value" for a given "input."

It is easy to start listing examples of 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 ..

Some examples follow:

1. **"One plus two equals three."**

   Objects: one, two, three, one plus two

   Relation: equals

   Function: plus
2. **"Squares neighboring the wumpus are smelly."**

   Objects: wumpus, squares

   Property: smelly

   Relation: neighboring.
3. **"Evil King John ruled England in 1200."**

   Objects: John, England, 1200

Relation: ruled

Properties: evil, king.

Syntax and Semantics of first-order logic

Models for first-order logic

Models for first-order logic have objects in them.

The domain of a model is the set of objects or domain elements it contains.

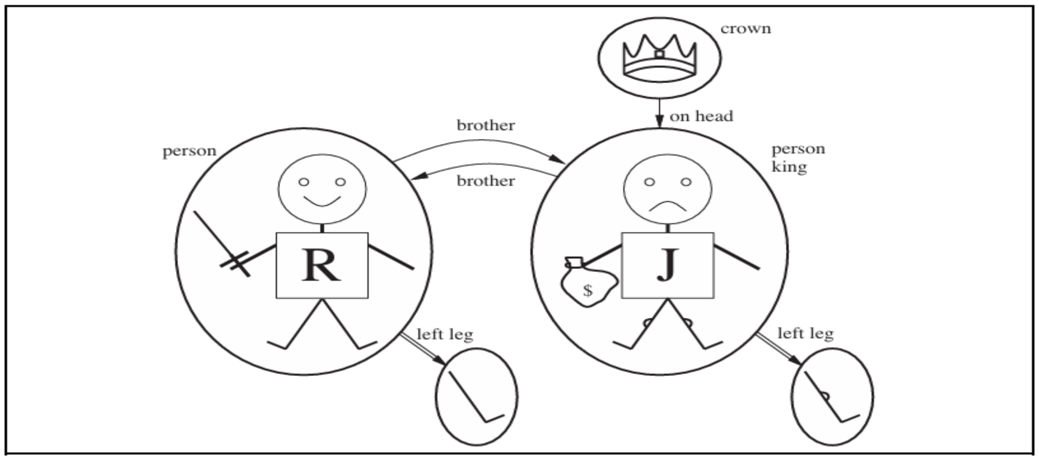


Figure 3: A model containing five objects, two binary relations, three unary relations (indicated by labels on the objects), and one unary function, left-leg.

Above figure 3 shows a model with **five objects**:

- Richard the Lionheart, King of England from 1189 to 1199;
- his younger brother, the evil King John, who ruled from 1199 to 1215;
- the left legs of Richard and John;
- and a crown.

The objects in the model may be ***related*** in various ways - **Richard and John are brothers.**

- relation is just the set of tuples of objects that are related.
- brotherhood relation in this model is the set
- { <Richard the Lionheart, King John> , <King John, Richard the Lionheart> }
- The crown is on King John's head
- so the "on head" relation contains just one tuple
- <the crown, King John>

- “brother” and “on head” relations are binary relations—that is, they relate pairs of objects

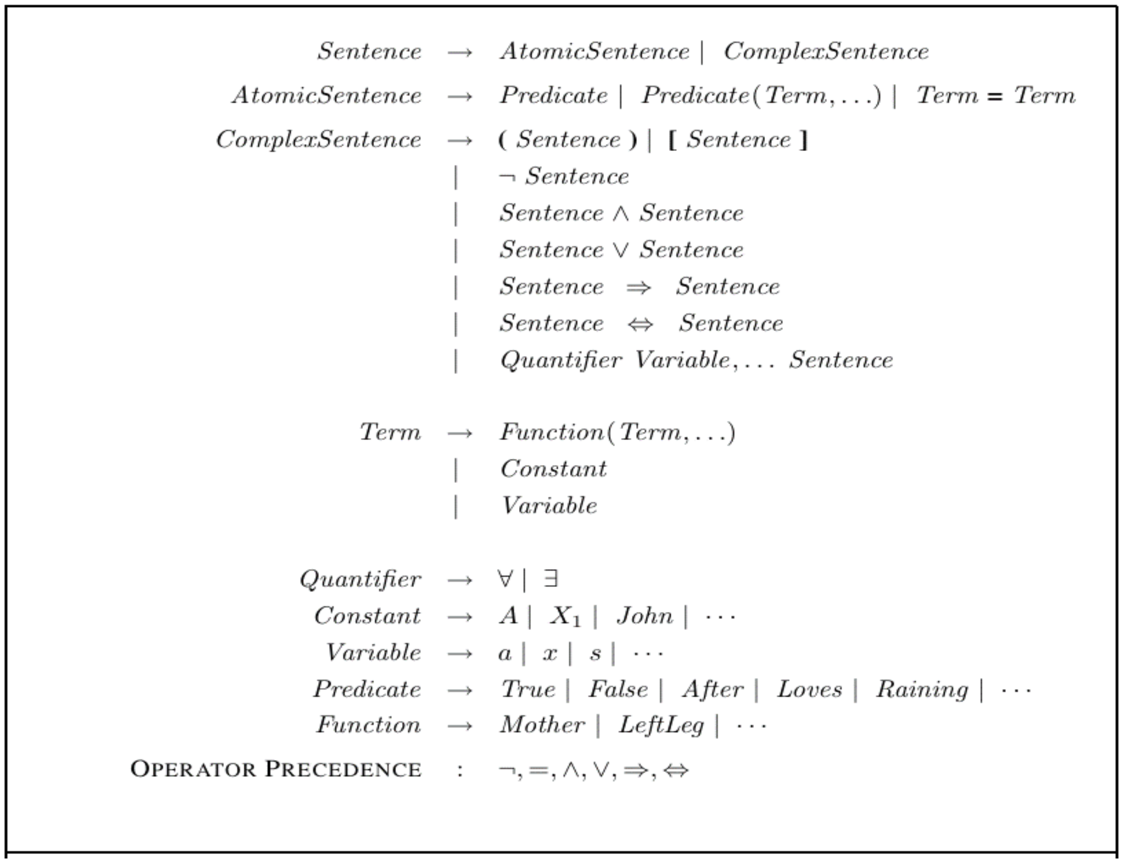
The model also contains **unary relations, or properties:**

- the “person” property is true of both Richard and John;
- the “king” property is true only of John;
- the “crown” property is true only of John.

relationships are best considered as **functions**:

- given object must be related to exactly one object
- **For example**, each person has one left leg, so the model has a unary “left leg” function:-
- <Richard the Lionheart> → Richard’s left leg
- <King John> → John’s left leg.

Symbols and interpretations



$Sentence$	$\rightarrow$	$AtomicSentence \mid ComplexSentence$
$AtomicSentence$	$\rightarrow$	$Predicate \mid Predicate(Term, \ldots) \mid Term = Term$
$ComplexSentence$	$\rightarrow$	$(\ Sentence\ ) \mid [\ Sentence\ ]$
	$\mid$	$\neg\ Sentence$
	$\mid$	$Sentence \wedge Sentence$
	$\mid$	$Sentence \vee Sentence$
	$\mid$	$Sentence \Rightarrow Sentence$
	$\mid$	$Sentence \Leftrightarrow Sentence$
	$\mid$	$Quantifier\ Variable, \ldots\ Sentence$
$Term$	$\rightarrow$	$Function(Term, \ldots)$
	$\mid$	$Constant$
	$\mid$	$Variable$
$Quantifier$	$\rightarrow$	$\forall \mid \exists$
$Constant$	$\rightarrow$	$A \mid X_1 \mid John \mid \cdots$
$Variable$	$\rightarrow$	$a \mid x \mid s \mid \cdots$
$Predicate$	$\rightarrow$	$True \mid False \mid After \mid Loves \mid Raining \mid \cdots$
$Function$	$\rightarrow$	$Mother \mid LeftLeg \mid \cdots$
OPERATOR PRECEDENCE	:	$\neg, =, \wedge, \vee, \Rightarrow, \Leftrightarrow$

Figure 3.1: The syntax of first-order logic with equality, specified in Backus–Naur form

- The symbols that stand for objects, relations, and functions. The symbols, come in three kinds:

- **constant symbols -** which stand for objects;
- **predicate symbols -** which stand for relations
- **function symbols -** which stand for functions

Note: these symbols will begin with uppercase letters. (Richard and John, Brother….)

- Each predicate and function symbol comes with an arity that fixes the number of arguments.
- As in propositional logic, every model must provide the information required to determine if any given sentence is true or false.
- Each model includes an interpretation that specifies exactly which objects, relations and functions are referred to by the constant, predicate, and function symbols.

Terms

A term is a logical expression that refers to an object.

Atomic sentences

- An atomic sentence is formed from a predicate symbol optionally followed by a parenthesized list of terms such as

  Brother (Richard, John).
- Atomic sentences can have complex terms as arguments
- Married (Father (Richard), Mother (John))

Complex sentences

use **logical connectives** to construct complex sentences

- ¬ Brother (LeftLeg (Richard), John)
- Brother (Richard, John) ∧ Brother (John, Richard)
- King (Richard) ∨ King (John)
- ¬ King (Richard) ⇒ King (John)

Quantifiers

First-order logic contains two standard quantifiers, called ***universal*** **and *existential***

1. **<u>Universal quantification (∀)</u>** - " For all "
   - "All kings are persons," is written in first-order logic as

     $\forall$ x King (x) $\Rightarrow$ Person (x) .
   - For all x, if x is a king, then x is a person.
   - The symbol x is called a variable. Variables are lowercase letters
   - A variable is a term all by itself and can also serve as the argument of a function
   - For example, LeftLeg (x)
   - Term with no variables is called a **ground term.**

2. **<u>Existential quantification (∃)</u>** –
   - 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.
   - King John has a crown on his head

     $\exists$ x Crown (x) $\wedge$ OnHead (x, John)
   - $\exists$x is pronounced " There exists an x such that ..." or "For some x..."
   - $\Rightarrow$ is the natural connective to use with $\forall$,
   - $\wedge$ is the natural connective to use with $\exists$.

<u>Nested quantifiers</u>

Can express more complex sentences using multiple quantifiers. The simplest case is where the quantifiers are of the same type.

- For example, "Brothers are siblings"

  $\forall$ x $\forall$ y Brother (x, y) $\Rightarrow$ Sibling (x, y)
- Consecutive quantifiers of the same type can be written as one quantifier with several variables.
- For example, siblinghood is a symmetric relationship

  $\forall$ x, y Sibling (x, y) $\Leftrightarrow$ Sibling (y, x)

In other cases we will have **mixture of quantifiers**

- “Everybody loves somebody” means that for every person, there is someone that person loves: ∀ x ∃ y Loves (x, y) .
- “There is someone who is loved by everyone”

  ∃ y ∀ x Loves (x, y)
- The order of quantification is therefore very important
- ∀ x (∃ y Loves (x, y)) says that *everyone* has a particular property, namely, the property that they love someone.
- ∃ y (∀ x Loves (x, y)) says that *someone* in the world has a particular property, namely the property of being loved by everybody.

Connections between ∀ and ∃

The two quantifiers are actually intimately connected with each other, through **negation.**

∀ x ¬Likes (x, Parsnips) is equivalent to ¬∃ x Likes (x, Parsnips)

“Everyone likes ice cream” means that there is no one who does not like ice cream:

∀ x Likes (x, IceCream) is equivalent to ¬ ∃ x ¬ Likes (x, IceCream)

The **De Morgan rules** for quantified and unquantified sentences are as follows

$$
\begin{array}{lcl@{\qquad\qquad}lcl}
\forall x\ \neg P & \equiv & \neg\exists x\ P & \neg(P \lor Q) & \equiv & \neg P \land \neg Q \\
\neg\forall x\ P & \equiv & \exists x\ \neg P & \neg(P \land Q) & \equiv & \neg P \lor \neg Q \\
\forall x\ P & \equiv & \neg\exists x\ \neg P & P \land Q & \equiv & \neg(\neg P \lor \neg Q) \\
\exists x\ P & \equiv & \neg\forall x\ \neg P & P \lor Q & \equiv & \neg(\neg P \land \neg Q) .
\end{array}
$$

Equality

The equality symbol is used to signify that two terms refer to the same object.

For example,

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**

∃ x, y Brother (x, Richard) ∧ Brother (y, Richard) ∧ ¬ (x = y)

Using first-order logic

Assertions and queries in first-order logic

- Sentences are added to a knowledge base using TELL. Such sentences are called **assertions.**

  TELL (KB, sentence)

- For example, we can assert that John is a king, Richard is a person, and all kings are persons:

  TELL (KB, King (John))

  TELL (KB, Person (Richard))

  TELL (KB, $\forall$ x King (x) $\Rightarrow$ Person (x))

- We can ask questions of the knowledge base using ASK.

  ASK (KB, King (John)) returns true

  ASK (KB, Person (John)) returns true

  ASK (KB, $\exists$x Person (x)) returns true

- Questions asked with ASK are called **queries or goals**

The 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.”

- The **objects** in our domain are people.
- 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.
- **functions** for Mother and Father, because every person has exactly one of each of these.

1. One’s mother is one’s female parent:

$\forall$ m, c Mother (c) = m $\Leftrightarrow$ Female (m) $\wedge$ Parent (m, c)

2. One’s husband is one’s male spouse:

$\forall$ w, h Husband (h, w) $\Leftrightarrow$ Male (h) $\wedge$ Spouse (h, w)

3. Male and female are disjoint categories:

$\forall$ x Male (x) $\Leftrightarrow \neg$ Female (x)

4. Parent and child are inverse relations:

$\forall$ p, c Parent (p, c) $\Leftrightarrow$ Child (c, p)

5. A grandparent is a parent of one's parent:

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

6. A sibling is another child of one's parents:

$\forall$ x, y Sibling (x, y) $\Leftrightarrow$ x $\neq$ y $\wedge \exists$ p Parent (p, x) $\wedge$ Parent (p, y)

Numbers, sets, and lists

- We need a predicate **NatNum** that will be true of natural numbers; we need one constant symbol, 0, and we need one function symbol, S (successor).
- Natural numbers are defined recursively:

  NatNum(0) .

  $\forall$ n NatNum (n) $\Rightarrow$ NatNum (S(n))
- We also need axioms to constrain the successor function:

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

  $\forall$ m, n m $\neq$ n $\Rightarrow$ S(m) $\neq$ S(n).
- define addition in terms of the successor function:

  $\forall$ m NatNum(m) $\Rightarrow$ + (0, m) = m .

  $\forall$ m, n NatNum(m) $\wedge$ NatNum(n) $\Rightarrow$ + (S(m), n) = S(+ (m, n))

SETS

The use of infix notation is an example of **syntactic sugar**; 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 $\in$ s (x is a member of set s) and s1 $\subseteq$ s2 (set s1 is a subset)
- The **binary functions** are s1 $\cap$ s2 (the intersection of two sets), s1 $\cup$ s2 (the union of two sets), and {x | s} (the set resulting from adjoining element x to set s).

One possible set of axioms is as follows:

1. The only sets are the empty set and those made by adjoining something to a set:

$\forall s\ Set(s) \Leftrightarrow (s = \{\}) \lor (\exists x, s2\ Set(s2) \land s = \{x \mid s2\})$

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:

   $\neg \exists x, s\ \{x \mid s\} = \{\}$

3. Adjoining an element already in the set has no effect:

   $\forall x, s\ x \in s \Leftrightarrow s = \{x \mid s\}$

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:

   $\forall x, s\ x \in s \Leftrightarrow \exists y, s2\ (s = \{y \mid s2\} \land (x = y \lor x \in s2))$

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:

   $\forall s1, s2\ (s1 = s2) \Leftrightarrow (s1 \subseteq s2 \land s2 \subseteq s1)$

7. An object is in the intersection of two sets if and only if it is a member of both sets:

   $\forall x, s1, s2\ x \in (s1 \cap s2) \Leftrightarrow (x \in s1 \land x \in s2)$

8. An object is in the union of two sets if and only if it is a member of either set:

   $\forall x, s1, s2\ x \in (s1 \cup s2) \Leftrightarrow (x \in s1 \lor x \in s2)$.

<u>LISTS</u>

Lists are similar to sets. The differences are that lists are ordered and the same element can appear more than once in a list.

<u>The wumpus world</u>

The wumpus agent receives a percept vector with five elements:

Percept ([Stench, Breeze, Glitter , None, None], 5)

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}

Simple “reflex” behavior can also be implemented by quantified implication sentences

$\forall$ t Glitter (t) $\Rightarrow$ BestAction (Grab, t)

Knowledge engineering in first-order logic

A knowledge engineer is someone who investigates a particular domain, learns what concepts are important in that domain, and creates a formal representation of the objects and relations in the domain.

It includes the following steps:

1. **Identify the task: -** The task will determine what knowledge must be represented in order to connect problem instances to answers. This step is analogous to the PEAS process for designing agents.
2. **Assemble the relevant knowledge: -** The knowledge engineer might already be an expert in the domain, or might need to work with real experts to extract what they know—a process called knowledge acquisition.
3. **Decide on a vocabulary of predicates, functions, and constants: -** That is, translate the important domain-level concepts into logic-level names. Once the choices have been made, the result is a vocabulary that is known as the ontology of the domain. The word ontology means a particular theory of the nature of being or existence.
4. **Encode general knowledge about the domain: -** The knowledge engineer writes down the axioms for all the vocabulary terms. This pins down (to the extent possible) the meaning of the terms, enabling the expert to check the content. Often, this step reveals misconceptions or gaps in the vocabulary that must be fixed by returning to step 3 and iterating through the process.
5. **Encode a description of the specific problem instance: -** For a logical agent, problem instances are supplied by the sensors, whereas a “disembodied” knowledge base is sup plied with additional sentences in the same way that traditional programs are supplied with input data.
6. **Pose queries to the inference procedure and get answers: -** This is where the reward is: we can let the inference procedure operate on the axioms and problem-specific facts to derive the facts we are interested in knowing.
7. **Debug the knowledge base:-** the answers will be correct for the knowledge base as

written, assuming that the inference procedure is sound, but they will not be the ones that the user is expecting.

For example, the sentence $\forall x$ NumOfLegs (x, 4) $\Rightarrow$ Mammal (x) is false for reptiles, amphibians

INFERENCE IN FIRST-ORDER LOGIC

Propositional versus first-order inference

1) Inference rules for quantifiers

- Suppose our knowledge base contains the standard axiom stating that all greedy kings are evil:

  ∀ x King(x) ∧ Greedy(x) ⇒ Evil(x)

- Then it seems quite permissible to infer any of the following sentences:

  King (John) ∧ Greedy (John) ⇒ Evil (John)

  King (Richard) ∧ Greedy (Richard) ⇒ Evil (Richard)

  King (Father (John)) ∧ Greedy (Father (John)) ⇒ Evil (Father (John)) .

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.

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

In the rule for **Existential Instantiation,** the variable is replaced by a single new constant symbol.

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

- For example, from the sentence

  ∃ x Crown(x) ∧ OnHead (x, John)

- we can infer the sentence

Crown(C1) ∧ OnHead(C1, John)

C1 does not appear elsewhere in the knowledge base.

2) Reduction to propositional inference

- suppose our knowledge base contains just the sentences

  ∀ x King(x) ∧ Greedy(x) ⇒ Evil(x)

  King(John)

  Greedy(John)

  Brother (Richard, John)

- apply UI to the first sentence using all possible ground-term substitutions from {x/John} and {x/Richard}. We obtain

  King(John) ∧ Greedy(John) ⇒ Evil(John)

  King(Richard) ∧ Greedy(Richard) ⇒ Evil(Richard)

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.

"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 ColonelWest, who is American."

Prove that West is a criminal.

Represent these facts as first-order definite clauses:-

- "it is a crime for an American to sell weapons to hostile nations":

  American(x) ∧ Weapon(y) ∧ Sells (x, y, z) ∧ Hostile(z) ⇒ Criminal (x)

- 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)

- "All of its missiles were sold to it by Colonel West"

  $Missile(x) \land Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$.
- We will also need to know that missiles are weapons:

  $Missile(x) \Rightarrow Weapon(x)$
- and we must know that an enemy of America counts as "hostile":

  $Enemy(x, America) \Rightarrow Hostile(x)$
- "West, who is American . . ."

  American(West)
- "The country Nono, an enemy of America . . .":

  Enemy (Nono, America) .

Forward Chaining Algorithm: -

- $American(x) \land Weapon(y) \land Sells(x, y, z) \land Hostile(z) \Rightarrow Criminal(x)$
- Owns (Nono, M1)
- Missile(M1)
- $Missile(x) \land Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$
- $Missile(x) \Rightarrow Weapon(x)$
- $Enemy(x, America) \Rightarrow Hostile(x)$
- American(West)
- Enemy (Nono, America)

Step-1: -

Start with the known facts and will choose the sentences which do not have implications, such as:

- American(West)
- Enemy (Nono, America)
- Owns (Nono, M1)
- Missile(M1)

Step-2: -

We will see those facts which infer from available facts and with satisfied premises.

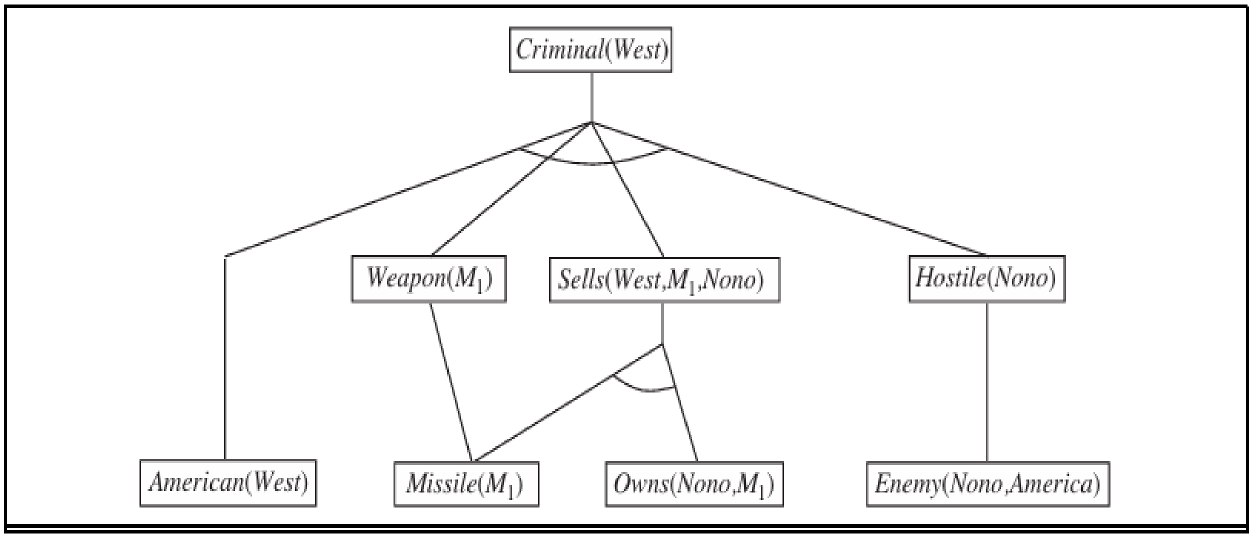


Figure 3.2: The proof tree generated by forward chaining on the crime example.

Resolution

1) Conjunctive normal form for first-order logic

- first-order resolution requires that sentences be in conjunctive normal form (CNF)
- a conjunction of clauses, where each clause is a disjunction of literals

"Everyone who loves all animals is loved by someone,"

First Order Logic: -

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

The steps are as follows: -

- Eliminate implications:
- Move ¬ inwards:
- Standardize variables:
- Skolemize:
- Drop universal quantifiers:
- Distribute ∨ over ∧:

- **Eliminate implications:**

  ∀x [¬∀ y ¬Animal(y) ∨ Loves(x, y)] ∨ [∃ y Loves(y, x)]

- **Move ¬ inwards:**

  ¬∀ x p becomes ∃ x ¬p

  ¬∃ x p becomes ∀ x ¬p

  ∀ 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:**

  ∀ x [∃ y Animal(y) ∧ ¬Loves(x, y)] ∨ [∃ z Loves(z, x)]

- **Skolemize**: process of removing existential quantifiers by elimination. translate ∃ x P(x) into P(A), where A is a new constant

  ∀ x [Animal(A) ∧ ¬Loves(x, A)] ∨ Loves(B,x)

  ∀ x [Animal(F(x)) ∧ ¬Loves(x, F(x))] ∨ Loves(G(z), x)

- **Drop universal quantifiers**:

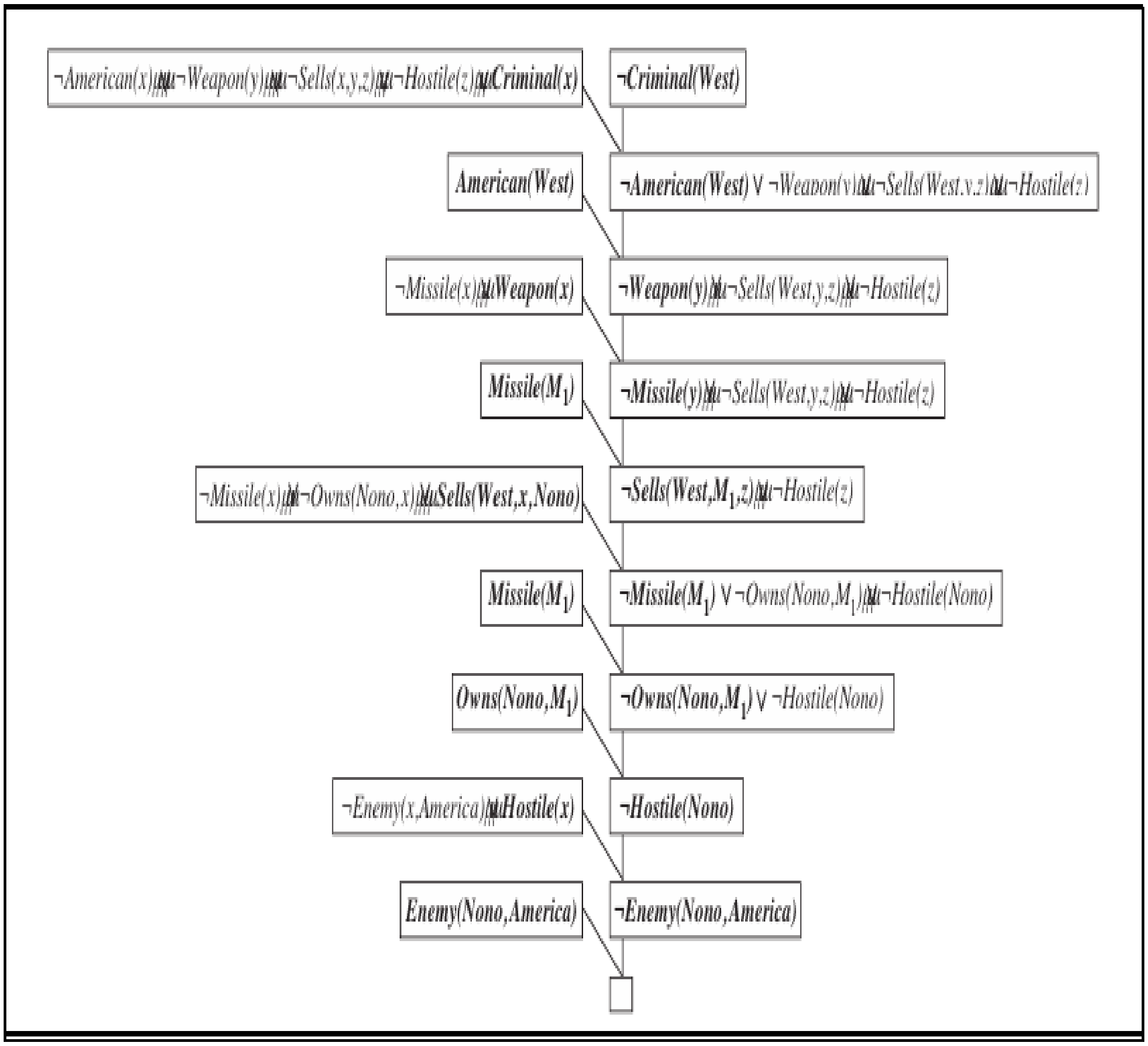
  [Animal(F(x)) ∧ ¬Loves(x, F(x))] ∨ Loves(G(z), x)

- **Distribute ∨ over ∧:**

  [Animal(F(x)) ∨ Loves(G(z), x)] ∧ [¬Loves(x, F(x)) ∨ Loves(G(z), x)]

2) The resolution inference rule

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


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