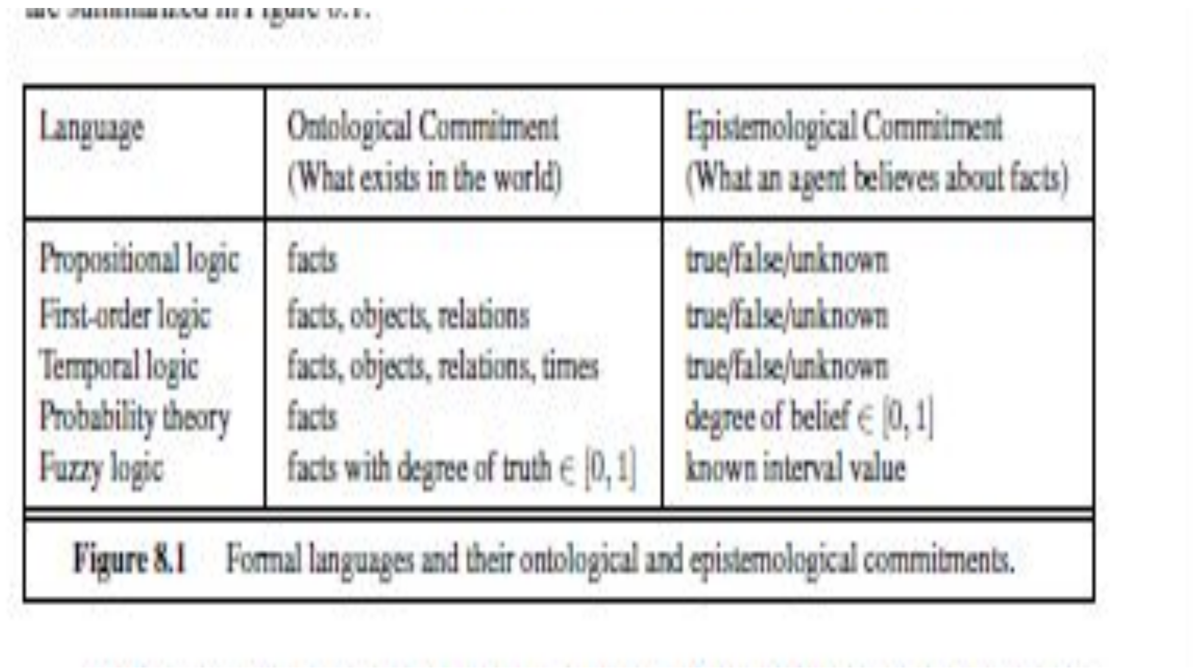
Module 4

Knowledge Representation

First-Order Logic (FOL)

- It is an extension to propositional logic.
- FOL is sufficiently expressive to represent the natural language statements in a concise way.
- First-order logic is also known as Predicate logic or First-order predicate logic



are summarized in Figure 8.1.

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

Figure 8.1 Formal languages and their ontological and epistemological commitments.

SYNTAX AND SEMANTICS OF FIRST-ORDER LOGIC

Models for first-order logic

- Following fig consists of 5 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.

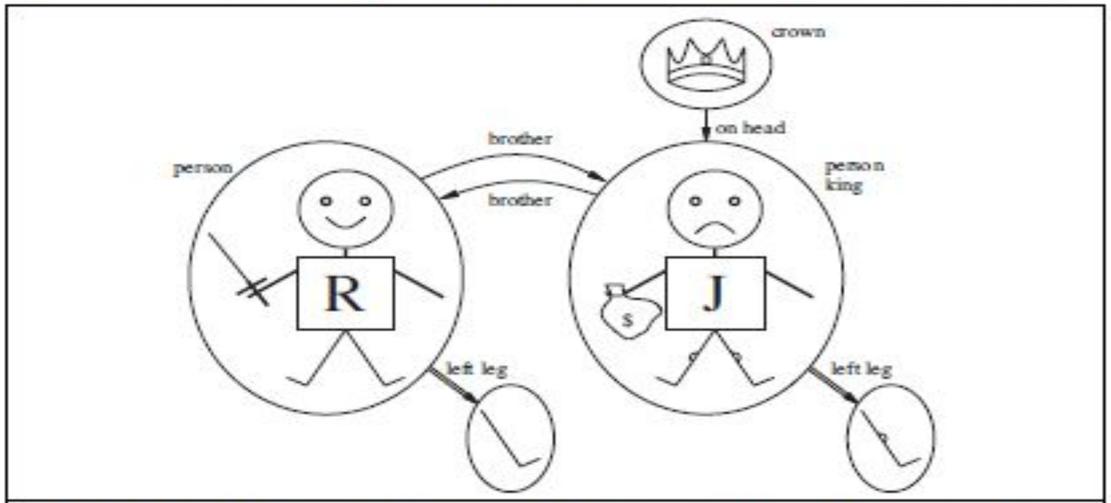


Figure 8.2 A model containing five objects, two binary relations, three unary relations (indicated by labels on the objects), and one unary function, left-leg.

- First-order logic (FOL) models the world in terms of

◦ **Objects,** which are things with individual identities

◦ **Properties** of objects that distinguish them from other objects

◦ **Relations** that hold among sets of objects

◦ **Functions,** which are a subset of relations where there is only one "value" for any given "input"

- Examples:
- Objects: Students, lectures…
- Relations: Brother-of, biggerthan, outside..
- Properties: blue, oval, even, large, ...
- Functions: father-of, best-friend, second-half, one-more-than ...

Symbols and interpretations

- The basic syntactic elements of first-order logic are the symbols that stand for objects,relations, and functions.
- The symbols are of three kinds:
  - **constant symbols**, which stand for objects;
  - **predicate symbols**, which stand for relations; and
  - **function symbols**, which stand for functions.
    - these symbols will begin with uppercase letters. For example, we might use the constant symbols Richard and John; the predicate symbols Brother , OnHead, Person, King, and Crown; and the function symbol LeftLeg.

Terms

- A **term** is a logical expression that refers to an object. Constant symbols are therefore terms, but it is not always convenient to have a distinct symbol to name every object.

Atomic sentences

- ATOMIC **sentence** (or **atom** for short) 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. Thus,

  Married(Father (Richard),Mother (John))

- *An atomic sentence is* ***true*** *in a given model if the relation referred to by the predicate symbol holds among the objects referred to by the arguments.*

Complex sentences

- use **logical connectives** to construct more complex sentences

  $\neg$Brother (LeftLeg(Richard), John)

  Brother (Richard , John) $\wedge$ Brother (John,Richard)

  King(Richard) $\vee$ King(John)

  $\neg$King(Richard) $\Rightarrow$ King(John)

Quantifiers

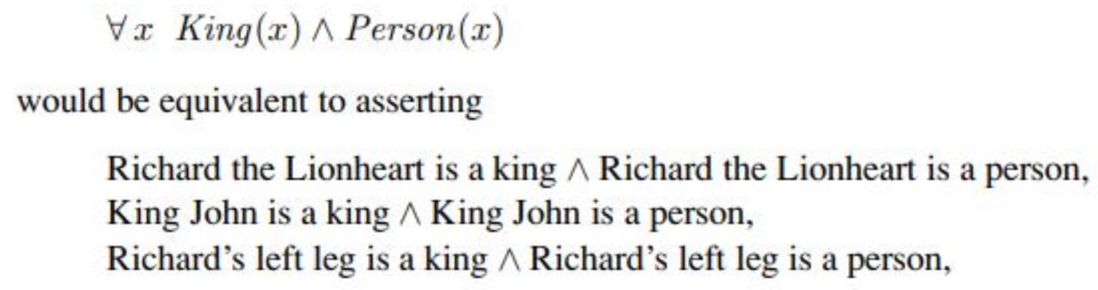
- two standard quantifiers, called *universal* and *existential*.

Universal quantification ($\forall$)

- expression of general rules in propositional logic
- The second rule, "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."

- $\forall$ is usually pronounced "For all ...".
- The symbol x is called a variable.
- 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.



$$\forall x \;\; King(x) \land Person(x)$$

would be equivalent to asserting

Richard the Lionheart is a king $\land$ Richard the Lionheart is a person,
King John is a king $\land$ King John is a person,
Richard's left leg is a king $\land$ Richard's left leg is a person,

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

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

- $\exists x$ is pronounced "There exists an x such that ..." or "For some x...".
- the sentence $\exists\, x\; P$ says that P is true for at least one object x.
- $\exists\, x\; P$ is true in a given model if P is true in at least one extended interpretation that assigns x to a domain element

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

Nested quantifiers

- We want to express more complex sentences using multiple quantifiers
- For example, “Brothers are siblings” can be written as

$$\forall\ x\ \forall\ y\ \text{Brother}(x, y) \Rightarrow \text{Sibling}(x, y)\ .$$

- Consecutive quantifiers of the same type can be written as one quantifier with several variables.
- For example, to say that siblinghood is a symmetric relationship, we can write

$$\forall\ x, y\ \text{Sibling}(x, y) \Leftrightarrow \text{Sibling}(y, x)\ .$$

- In other cases we will have mixtures. “Everybody loves somebody” means that for every person, there is someone that person loves:

$$\forall\ x\ \exists\ y\ \text{Loves}(x, y)\ .$$

- On the other hand, to say “There is someone who is loved by everyone,” we write

$$\exists\ y\ \forall\ x\ \text{Loves}(x, y)\ .$$

- $\forall$ x ($\exists$ y Loves(x, y)) says that everyone has a particular property, namely, the property that they love someone.
- On the other hand, $\exists$ y ($\forall$ x Loves(x, y)) says that someone in the world has a particular property, namely the property of being loved by everybody.

Connections between $\forall$ and $\exists$

- 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) .
- "Everyone likes ice cream" means that there is no one who does not like ice cream:

  $\forall$ x Likes(x,IceCream) is equivalent to ¬ $\exists$ x ¬Likes(x,IceCream)

De Morgan rules for quantified and unquantified sentences

$$\begin{array}{llll}
\forall x\ \neg P & \equiv \neg\exists x\ P & \qquad \neg(P \vee Q) & \equiv \neg P \wedge \neg Q \\
\neg\forall x\ P & \equiv \exists x\ \neg P & \qquad \neg(P \wedge Q) & \equiv \neg P \vee \neg Q \\
\forall x\ P & \equiv \neg\exists x\ \neg P & \qquad P \wedge Q & \equiv \neg(\neg P \vee \neg Q) \\
\exists x\ P & \equiv \neg\forall x\ \neg P & \qquad P \vee Q & \equiv \neg(\neg P \wedge \neg Q)\,.
\end{array}$$

Equality

- We can use the equality symbol 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, we would write

$$\exists x, y \; Brother(x, Richard) \wedge Brother(y, Richard) \wedge \neg(x = y) .$$

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

Figure 8.3 The syntax of first-order logic with equality, specified in Backus–Naur form (see page 1060 if you are not familiar with this notation). Operator precedences are specified, from highest to lowest. The precedence of quantifiers is such that a quantifier holds over everything to the right of it.

Examples of FOL

1. All birds fly.

- In this question the predicate is "**fly(bird)**."And since there are all birds who fly so it will be represented as follows.

$\forall$ **x bird(x) →fly(x)**.

2. Every man respects his parent.

- In this question, the predicate is "**respect(x, y)," where x=man, and y= parent**. Since there is every man so will use $\forall$, and it will be represented as follows:

$\forall$ **x man(x) → respects (x, parent)**.

3. Some boys play cricket.

- In this question, the predicate is "**play(x, y)**," where x= boys, and y= game. Since there are some boys so we will use $\exists$**, and it will be represented as**:

$\exists$ **x boys(x) → play(x, cricket)**.

4. Not all students like both Mathematics and Science.

- In this question, the predicate is "**like(x, y)," where x= student, and y=subject**.Since there are not all students, so we will use $\forall$ **with negation,so** following representation for this:

$\neg\forall$ (x) [student(x) → like(x, Mathematics) $\wedge$ like(x, Science)].

5. Only one student failed in Mathematics.

- In this question, the predicate is "**failed(x, y)," where x= student, and y=subject**.Since there is only one student who failed in Mathematics, so we will use following representation for this:

$\exists$(x) [student(x) → failed (x, Mathematics) $\wedge$ $\forall$ (y) [¬(x==y) $\wedge$ student(y) → ¬failed (x, Mathematics)].

8.10 Consider a vocabulary with the following symbols:

$Occupation(p, o)$: Predicate. Person p has occupation o.
$Customer(p1, p2)$: Predicate. Person $p1$ is a customer of person $p2$.
$Boss(p1, p2)$: Predicate. Person $p1$ is a boss of person $p2$.
$Doctor$, $Surgeon$, $Lawyer$, $Actor$: Constants denoting occupations.
$Emily$, Joe: Constants denoting people.

Use these symbols to write the following assertions in first-order logic:

a. Emily is either a surgeon or a lawyer.
b. Joe is an actor, but he also holds another job.
c. All surgeons are doctors.
d. Joe does not have a lawyer (i.e., is not a customer of any lawyer).
e. Emily has a boss who is a lawyer.
f. There exists a lawyer all of whose customers are doctors.
g. Every surgeon has a lawyer.

1. Emily is either a surgeon or a lawyer.

$$\text{Occupation(Emily, Surgeon)} \lor \text{ccupation(Emily, Lawyer)}$$

or

$$\text{Occupation(Emily, Surgeon)} \Leftrightarrow \text{Occupation(Emily, Lawyer)}$$

2. All surgeons are doctors.

$$\forall\ p\ [\text{Occupation}(p, \text{Surgeon}) \Rightarrow \text{Occupation}(p, \text{Doctor})]$$

3. Joe is an actor, but he holds another job.

$$\text{Occupation(Joe, Actor)} \land \exists\ o\ [\text{Occupation(Joe}, o) \land \neg (o = \text{Actor})]$$

or

$$\text{Occupation(Joe, Actor)} \land [\ \text{Occupation(Joe, Doctor)} \lor \text{Occupation(Joe, Surgeon)} \lor \text{Occupation(Joe, Lawyer)}\ ]$$

4. Joe does not have a lawyer (i.e., Joe is not a customer of any lawyer).

$$\forall\ p\ [\text{Occupation}(p, \text{Lawyer}) \Rightarrow \neg\ \text{Customer(Joe}, p)]$$

Or

$$\neg\ \exists\ p\ [\text{Occupation}(p, \text{Lawyer}) \land \text{Customer(Joe}, p)]$$

or

$$\forall\ p\ [\text{Customer(Joe}, p) \Rightarrow \neg\ \text{Occupation}(p, \text{Lawyer})]$$

5. Emily has a boss who is a lawyer.

$\exists$ p [Boss(p, Emily) $\wedge$ Occupation(p, Lawyer)]

6. Every surgeon has a lawyer (i.e., every surgeon is a customer of a lawyer).

$\forall$ p1 $\exists$ p2 Occupation(p1, Surgeon) $\Rightarrow$ [Customer(p1, p2) $\wedge$ Occupation(p2, Lawyer)]

Or

$\forall$ p1 Occupation(p1, Surgeon) $\Rightarrow$ [$\exists$ p2 Customer(p1, p2) $\wedge$ Occupation(p2, Lawyer)]

8.19 Assuming predicates Parent(p, q) and Female(p) and constants Joan and Kevin, with the obvious meanings, express each of the following sentences in first-order logic. (You may use the abbreviation ∃1 to mean "there exists exactly one.")

a. Joan has a daughter (possibly more than one, and possibly sons as well).

b. Joan has exactly one daughter (but may have sons as well).

c. Joan has exactly one child, a daughter.

d. Joan and Kevin have exactly one child together.

e. Joan has at least one child with Kevin, and no children with anyone else.

a) ∃x: Parent(Joan, x) ^ Female(x)
b) ∃1x: Parent(Joan, x) ^ Female(x)
c) ∃1x: Parent(Joan, x) -> Female(x)
d) ∃1x: Parent(Joan, x) ^ Parent(Kevin, x)
e) ∃1x: Parent(Joan, x) -> Parent(Kevin, x)

Free and Bound Variable

- There are two types of variables in First-order logic which are given below:
- Free Variable: A variable is said to be a free variable in a formula if it occurs outside the scope of the quantifier.

Example: $\forall x \exists (y)[P (x, y, z)]$, where z is a free variable.

- Bound Variable: A variable is said to be a bound variable in a formula if it occurs within the scope of the quantifier.

Example: $\forall x [A (x) B(y)]$, here x and y are the bound variables.

Unification

- The process of finding a substitution for predicate parameters is called *unification*.
- We need to know:

◦ that 2 literals can be matched.

◦ the substitution is that makes the literals identical.

- There is a simple algorithm called the *unification algorithm* that does this.

Conditions for Unification:

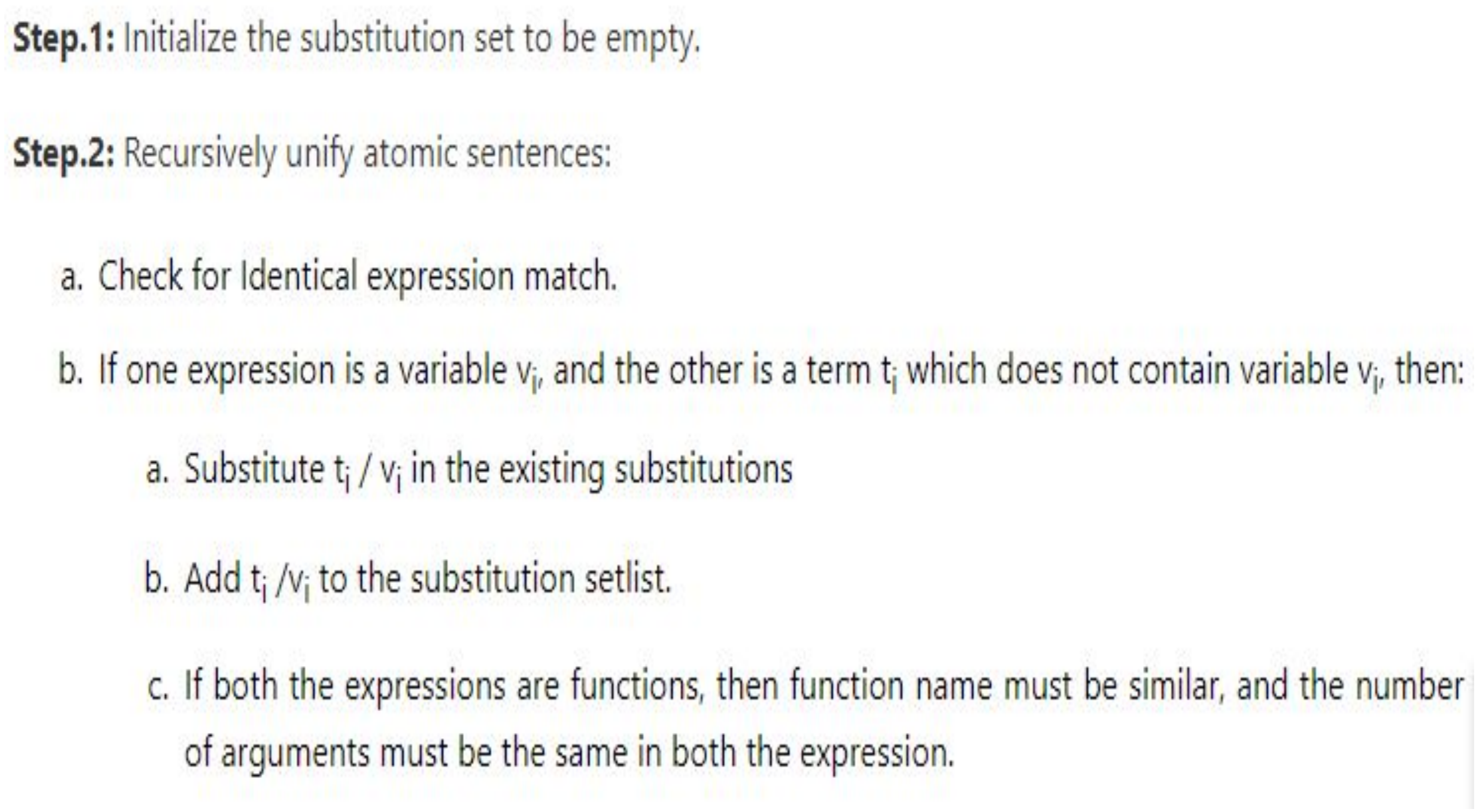
- **Following are some basic conditions for unification:**

•Predicate symbol must be same, atoms or expression with different predicate symbol can never be unified.

•Number of Arguments in both expressions must be identical.

•Unification will fail if there are two similar variables present in the same expression.

The Unification Algorithm



Step.1: Initialize the substitution set to be empty.

Step.2: Recursively unify atomic sentences:

a. Check for Identical expression match.

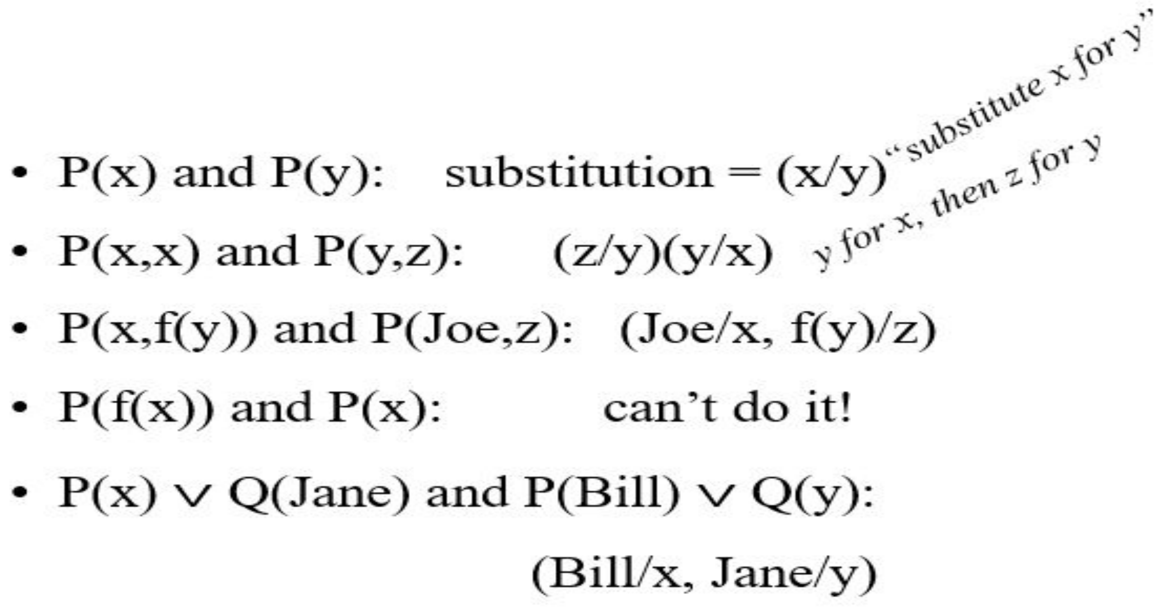
b. If one expression is a variable v_i, and the other is a term t_i which does not contain variable v_i, then:

  a. Substitute t_i / v_i in the existing substitutions

  b. Add t_i /v_i to the substitution setlist.

  c. If both the expressions are functions, then function name must be similar, and the number of arguments must be the same in both the expression.

Unification Example



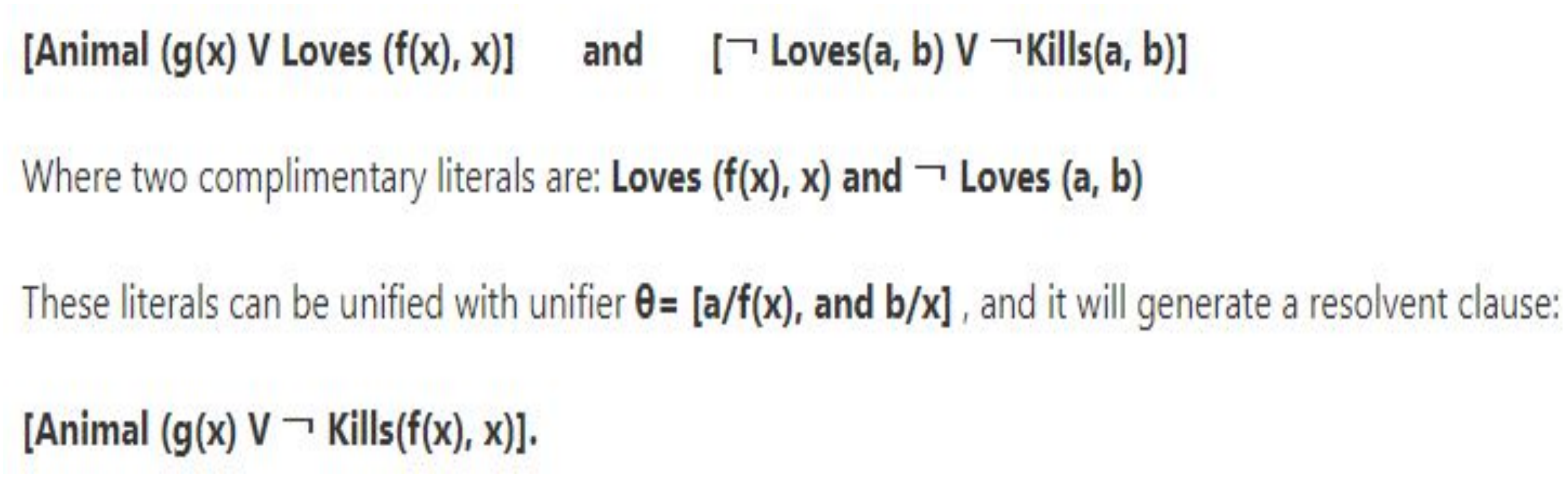
- P(x) and P(y): substitution = (x/y) “substitute x for y”
- P(x,x) and P(y,z): (z/y)(y/x) y for x, then z for y
- P(x,f(y)) and P(Joe,z): (Joe/x, f(y)/z)
- P(f(x)) and P(x): can’t do it!
- P(x) ∨ Q(Jane) and P(Bill) ∨ Q(y):

  (Bill/x, Jane/y)

Let Ψ1 = King(x), Ψ2 = King(John),
Substitution θ = {John/x} is a unifier for these atoms and applying this substitution, and both expressions will be identical

Resolution

- Resolution is a single inference rule which can efficiently operate on the **conjunctive normal form or clausal form**.
- **Example**



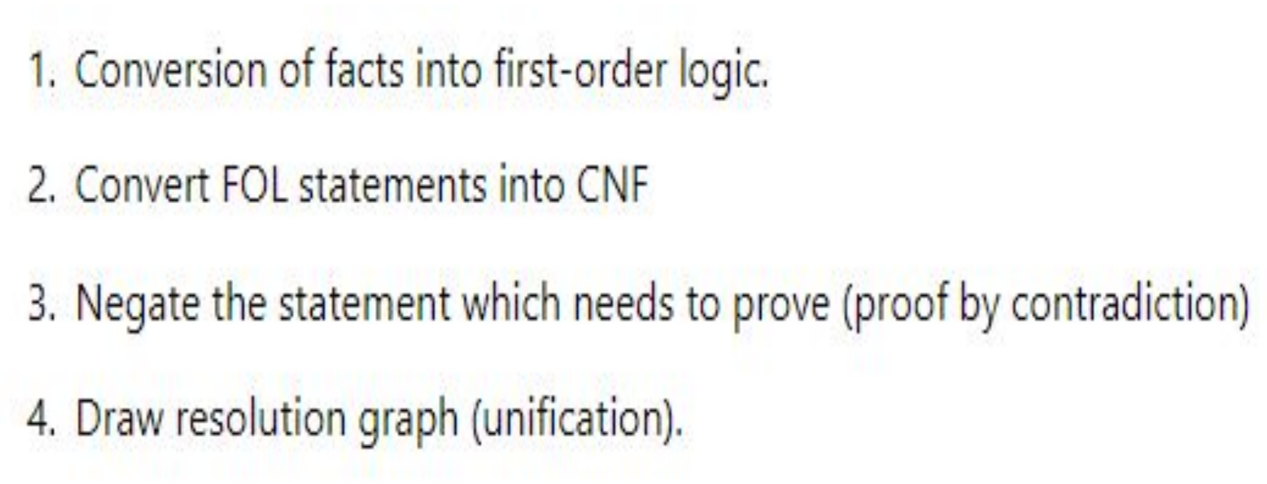
[Animal (g(x) V Loves (f(x), x)] **and** **[¬ Loves(a, b) V ¬Kills(a, b)]**

Where two complimentary literals are: **Loves (f(x), x) and ¬ Loves (a, b)**

These literals can be unified with unifier **θ= [a/f(x), and b/x]** , and it will generate a resolvent clause:

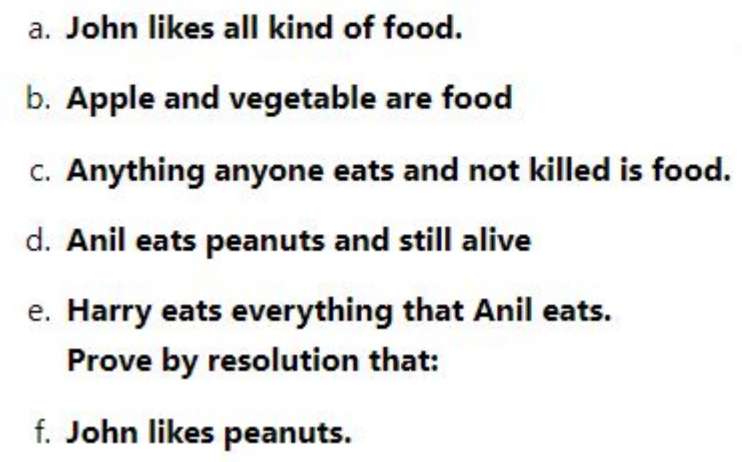
[Animal (g(x) V ¬ Kills(f(x), x)].

Steps for resolution



1. Conversion of facts into first-order logic.
2. Convert FOL statements into CNF
3. Negate the statement which needs to prove (proof by contradiction)
4. Draw resolution graph (unification).

Resolution- Example



a. **John likes all kind of food.**

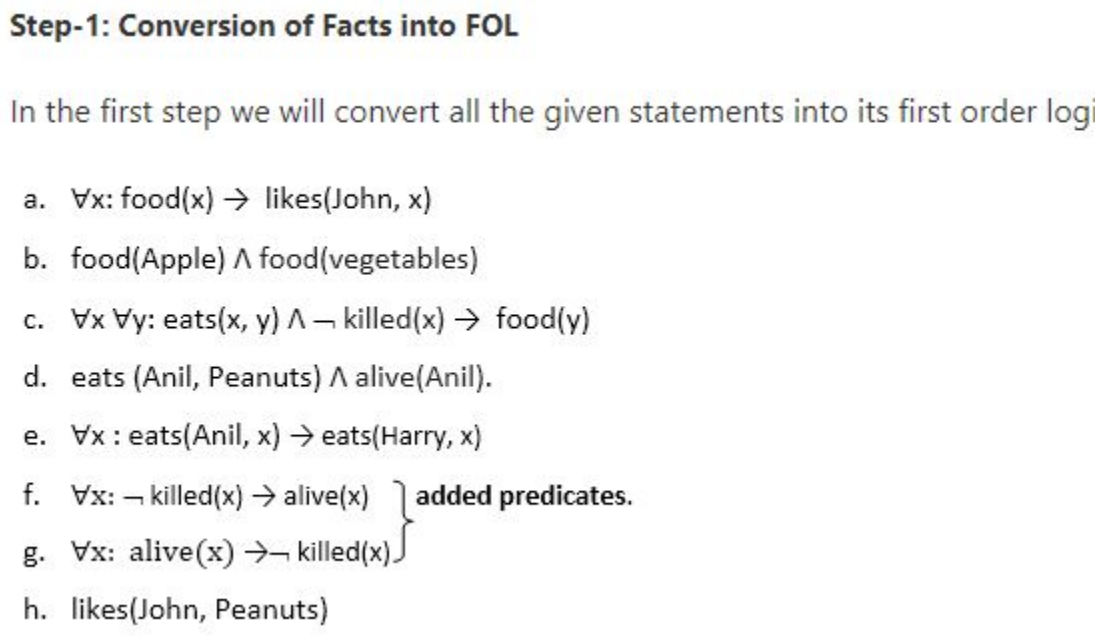
b. **Apple and vegetable are food**

c. **Anything anyone eats and not killed is food.**

d. **Anil eats peanuts and still alive**

e. **Harry eats everything that Anil eats.**
   Prove by resolution that:

f. **John likes peanuts.**



Step-1: Conversion of Facts into FOL

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

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

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

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

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

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

f. $\forall x$: $\neg$ killed(x) $\rightarrow$ alive(x) } **added predicates.**

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

h. likes(John, Peanuts)

Step-2: Conversion of FOL into CNF

In First order logic resolution, it is required to convert the FOL into CNF as CNF form makes easier for resolution proofs.

- **Eliminate all implication (→) and rewrite**
  a. ∀x ¬ food(x) V likes(John, x)
  b. food(Apple) Λ food(vegetables)
  c. ∀x ∀y ¬ [eats(x, y) Λ ¬ killed(x)] V food(y)
  d. eats (Anil, Peanuts) Λ alive(Anil)
  e. ∀x ¬ eats(Anil, x) V eats(Harry, x)
  f. ∀x¬ [¬ killed(x)] V alive(x)
  g. ∀x ¬ alive(x) V ¬ killed(x)
  h. likes(John, Peanuts).

- **Move negation (¬)inwards and rewrite**

  a. ∀x ¬ food(x) V likes(John, x)

  b. food(Apple) Λ food(vegetables)

  c. ∀x ∀y ¬ eats(x, y) V killed(x) V food(y)

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

  e. ∀x ¬ eats(Anil, x) V eats(Harry, x)

  f. ∀x ¬killed(x)] V alive(x)

  g. ∀x ¬ alive(x) V ¬ killed(x)

  h. likes(John, Peanuts).

$\longrightarrow$

- **Rename variables or standardize variables**

  a. ∀x ¬ food(x) V likes(John, x)

  b. food(Apple) Λ food(vegetables)

  c. ∀y ∀z ¬ eats(y, z) V killed(y) V food(z)

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

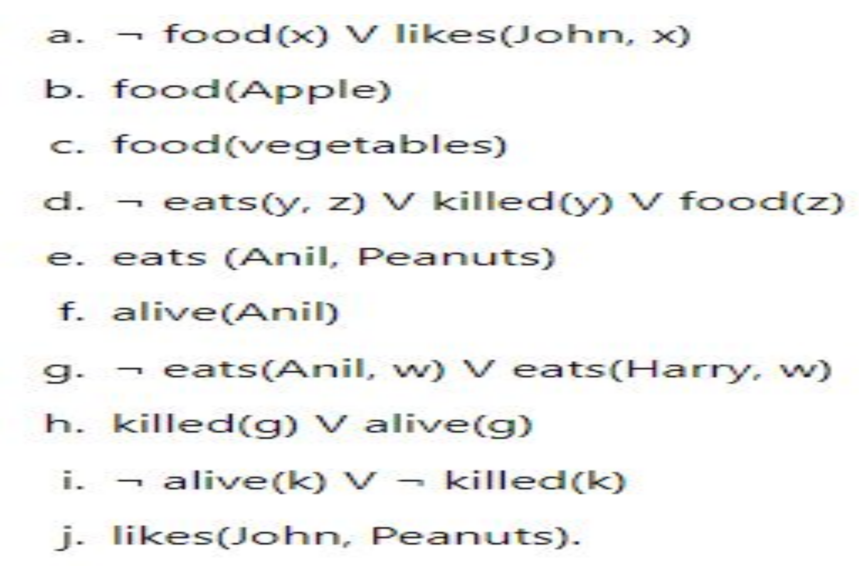
  e. ∀w¬ eats(Anil, w) V eats(Harry, w)

  f. ∀g ¬killed(g)] V alive(g)

  g. ∀k ¬ alive(k) V ¬ killed(k)

  h. likes(John, Peanuts).

Drop Universal Quantifier



a. ¬ food(x) V likes(John, x)

b. food(Apple)

c. food(vegetables)

d. ¬ eats(y, z) V killed(y) V food(z)

e. eats (Anil, Peanuts)

f. alive(Anil)

g. ¬ eats(Anil, w) V eats(Harry, w)

h. killed(g) V alive(g)

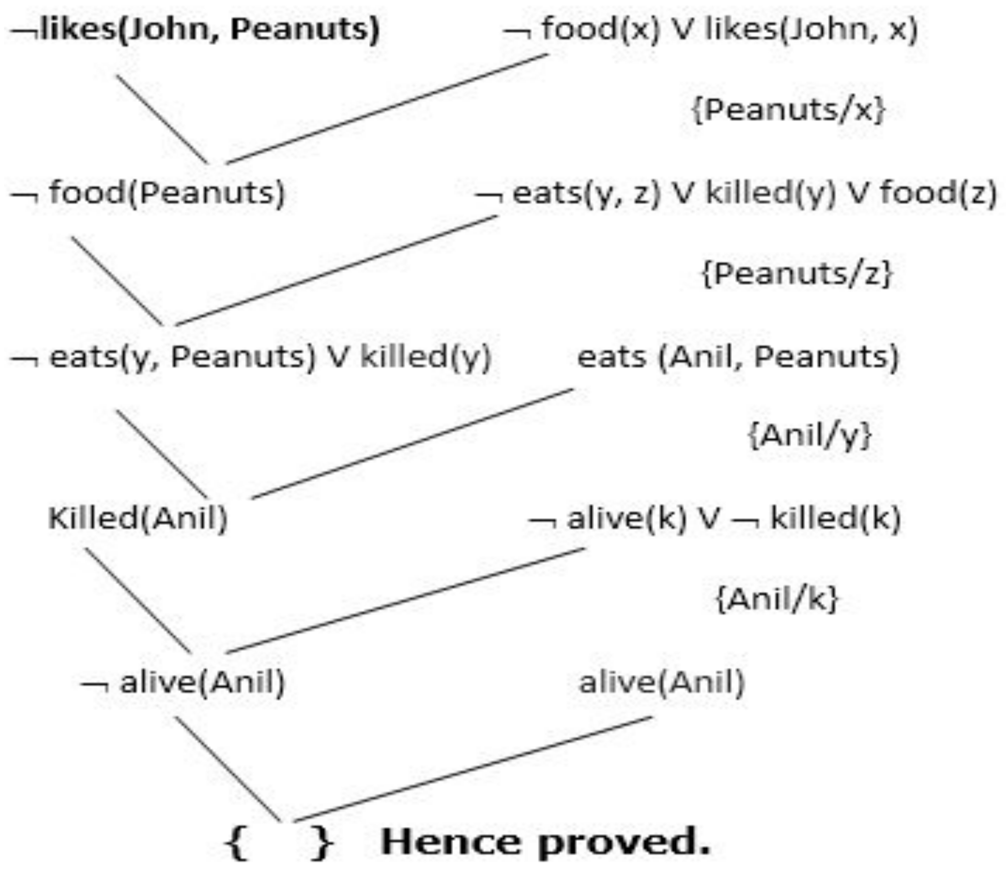
i. ¬ alive(k) V ¬ killed(k)

j. likes(John, Peanuts).

Step-3: Negate the statement to be proved

In this statement, we will apply negation to the conclusion statements, which will be written as ¬likes(John, Peanuts)

Step 4. Resolution Graph



Inference engine

- The inference engine is the component of the intelligent system in artificial intelligence, which applies logical rules to the knowledge base to infer new information from known facts.
- Inference engine commonly proceeds in two modes, which are:
  - **Forward chaining**
  - **Backward chaining**

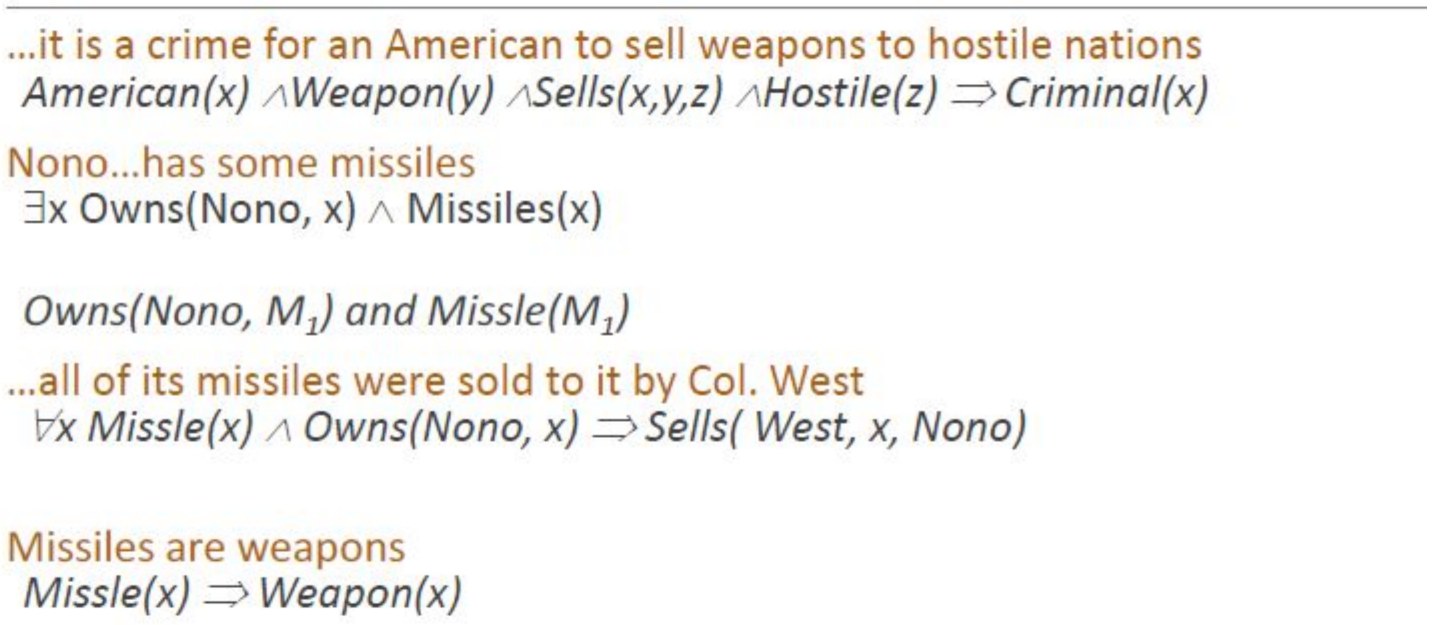
Forward Chaining

- Forward chaining is a form of reasoning which start with atomic sentences in the knowledge base and applies inference rules (Modus Ponens) in the forward direction to extract more data until a goal is reached.
- The Forward-chaining algorithm starts from known facts, triggers all rules whose premises are satisfied, and add their conclusion to the known facts. This process repeats until the problem is solved.

FC: Example Knowledge Base

- The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy America, has some missiles, and all of its missiles were sold to it by Col. West, who is an American.

  Prove that Col. West is a criminal.



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

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

Nono...has some missiles

∃x Owns(Nono, x) ∧ Missiles(x)

Owns(Nono, M_1) and Missle(M_1)

...all of its missiles were sold to it by Col. West

∀x Missle(x) ∧ Owns(Nono, x) ⇒ Sells(West, x, Nono)

Missiles are weapons

Missle(x) ⇒ Weapon(x)

An enemy of America counts as “hostile”

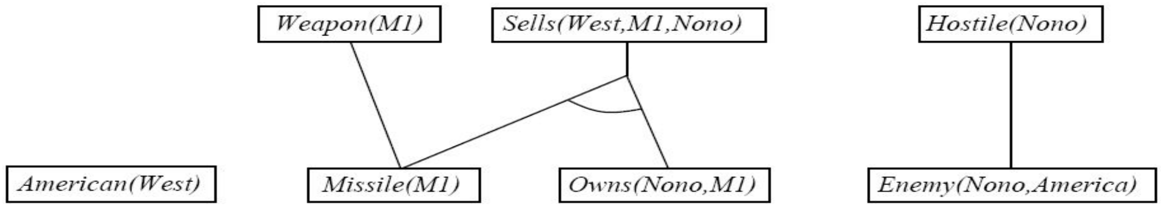
- *Enemy(x, America)* $\Rightarrow$ *Hostile(x)*

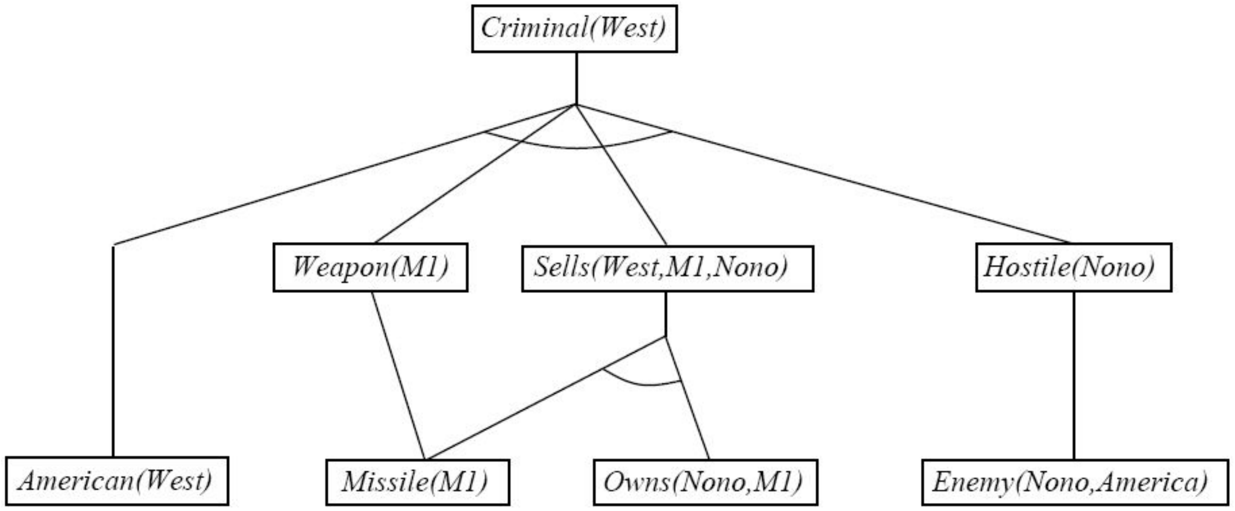
Col. West who is an American

- *American(Col. West)*

The country Nono, an enemy of America

- *Enemy(Nono, America)*



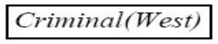


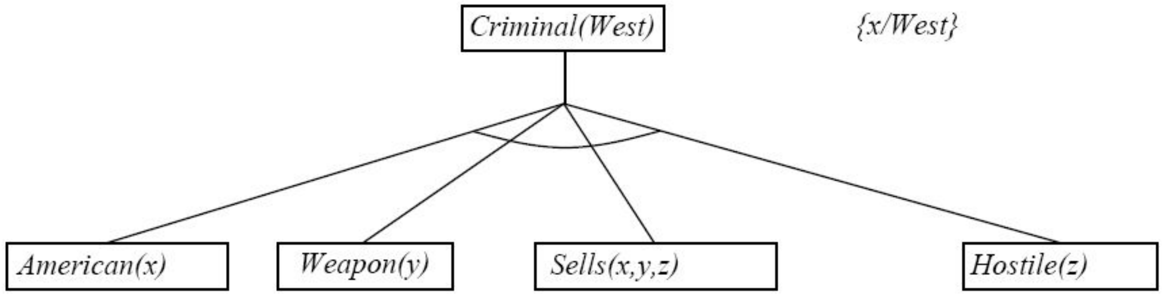
The proof tree generated by forward chaining on the crime example. The initial facts appear at the bottom level, facts inferred on the first iteration in the middle level, and facts inferred on the second iteration at the top level.

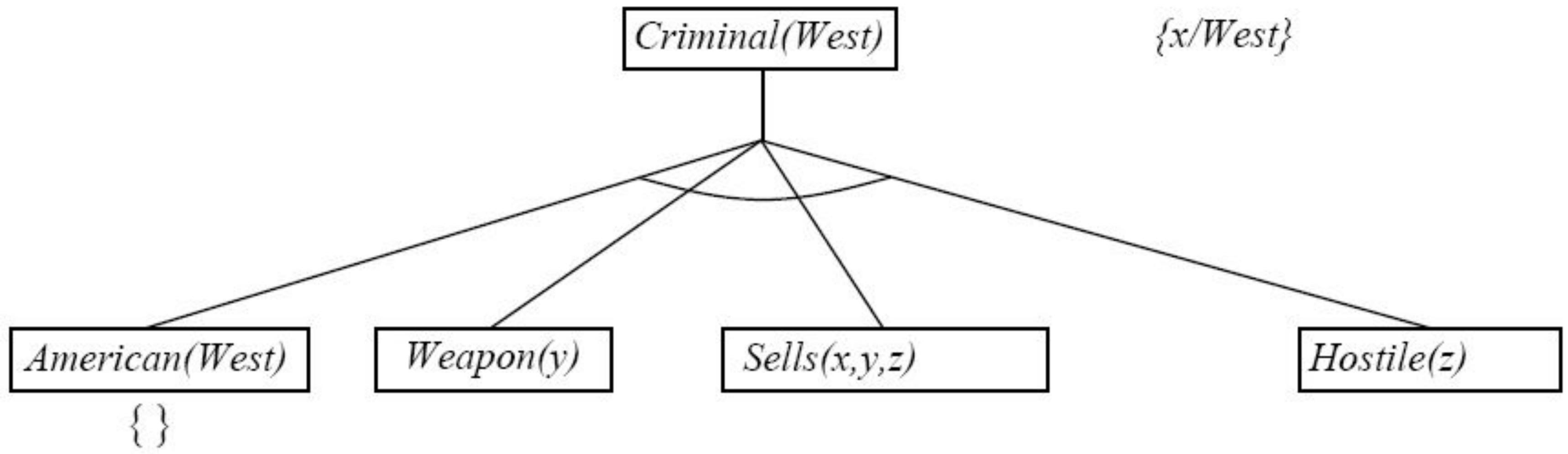
Backward Chaining

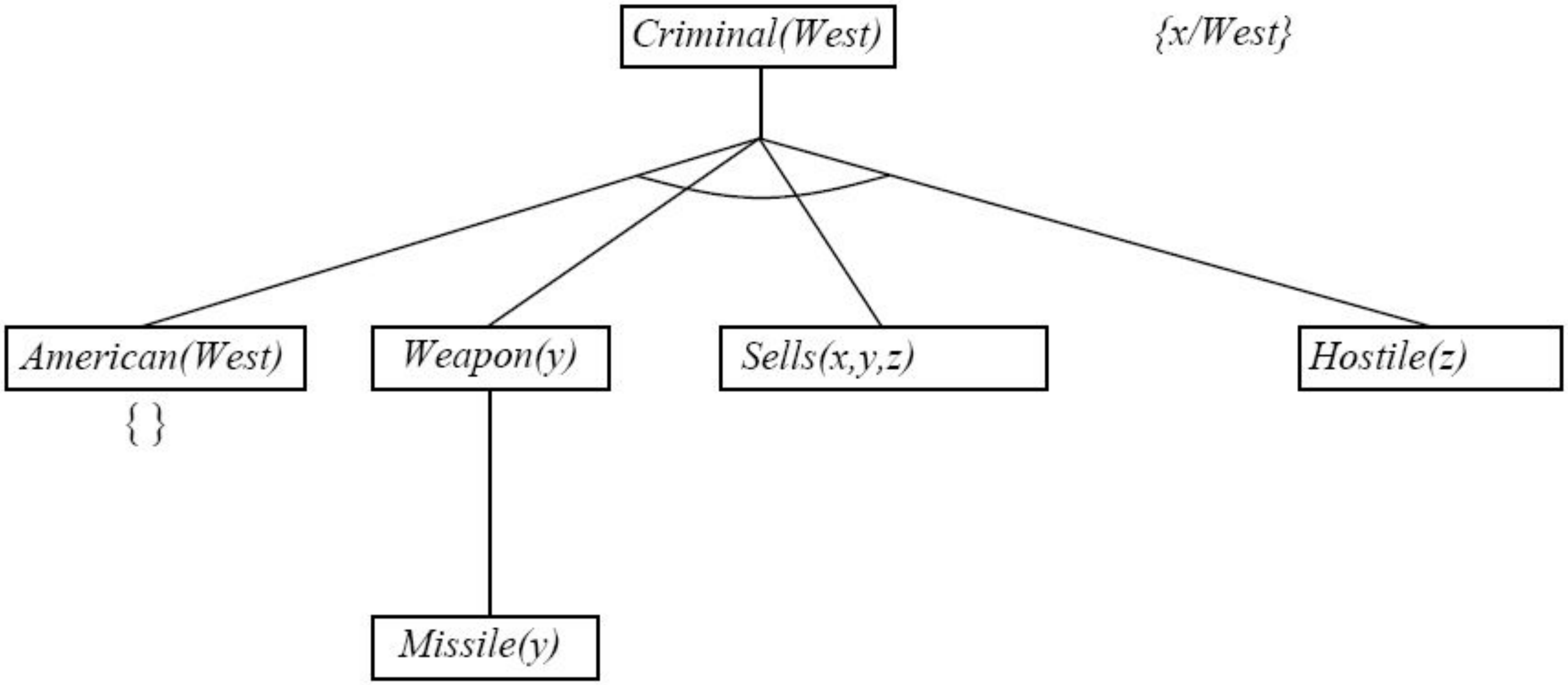
- Consider the item to be proven a goal
- Find a rule whose head is the goal (and bindings)
- Apply bindings to the body, and prove these (subgoals) in turn
- If you prove all the subgoals, increasing the binding set as you go, you will prove the item.

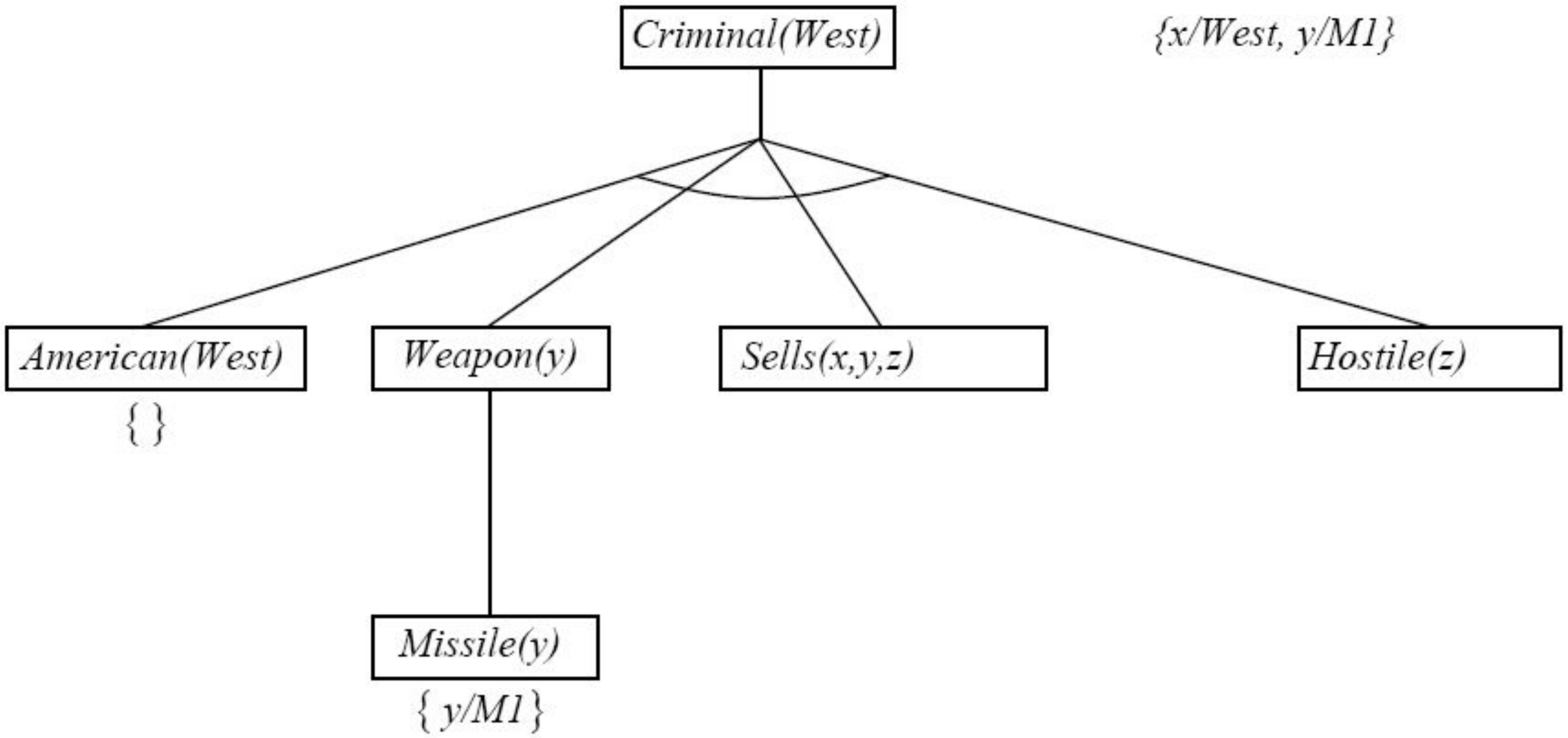
Backward Chaining Example

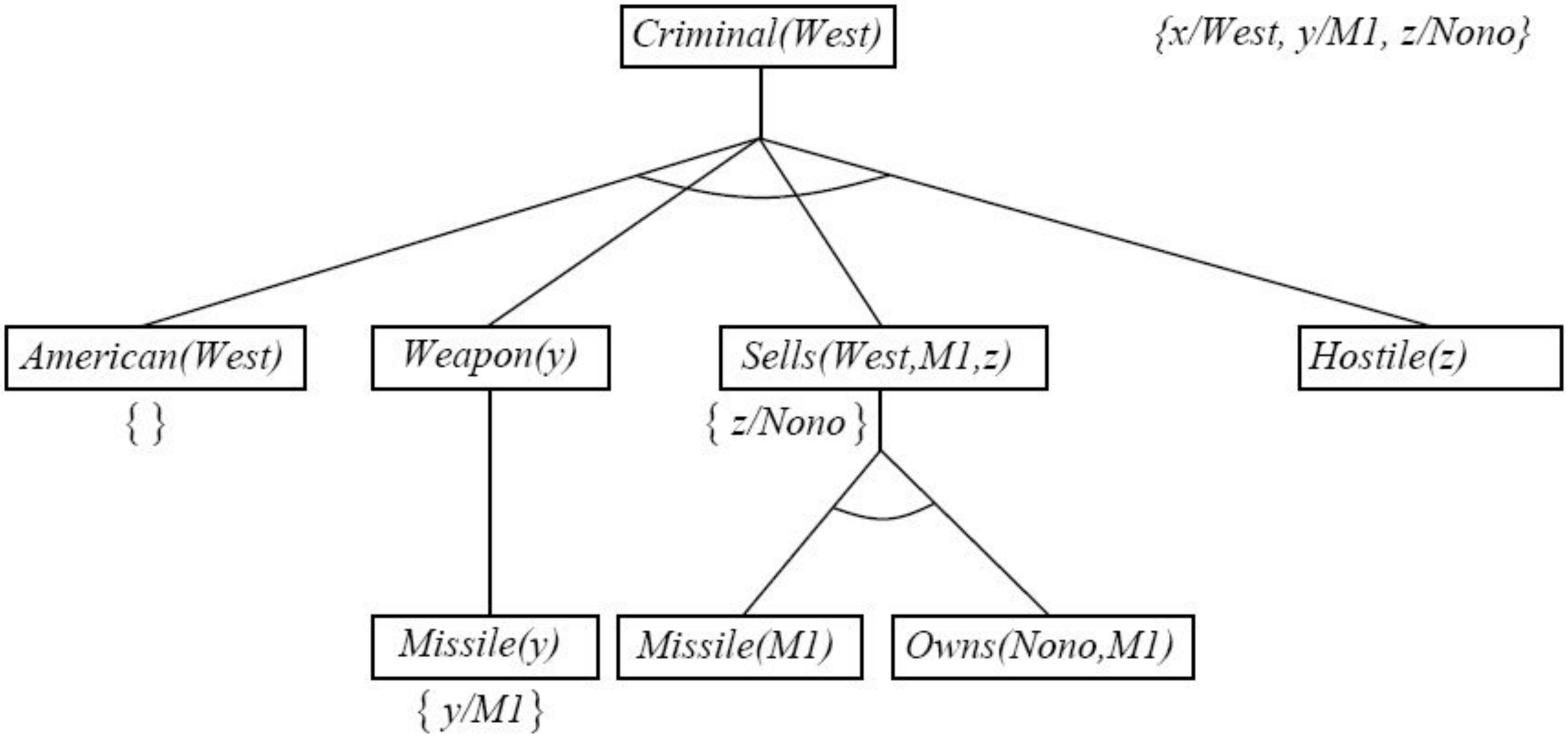


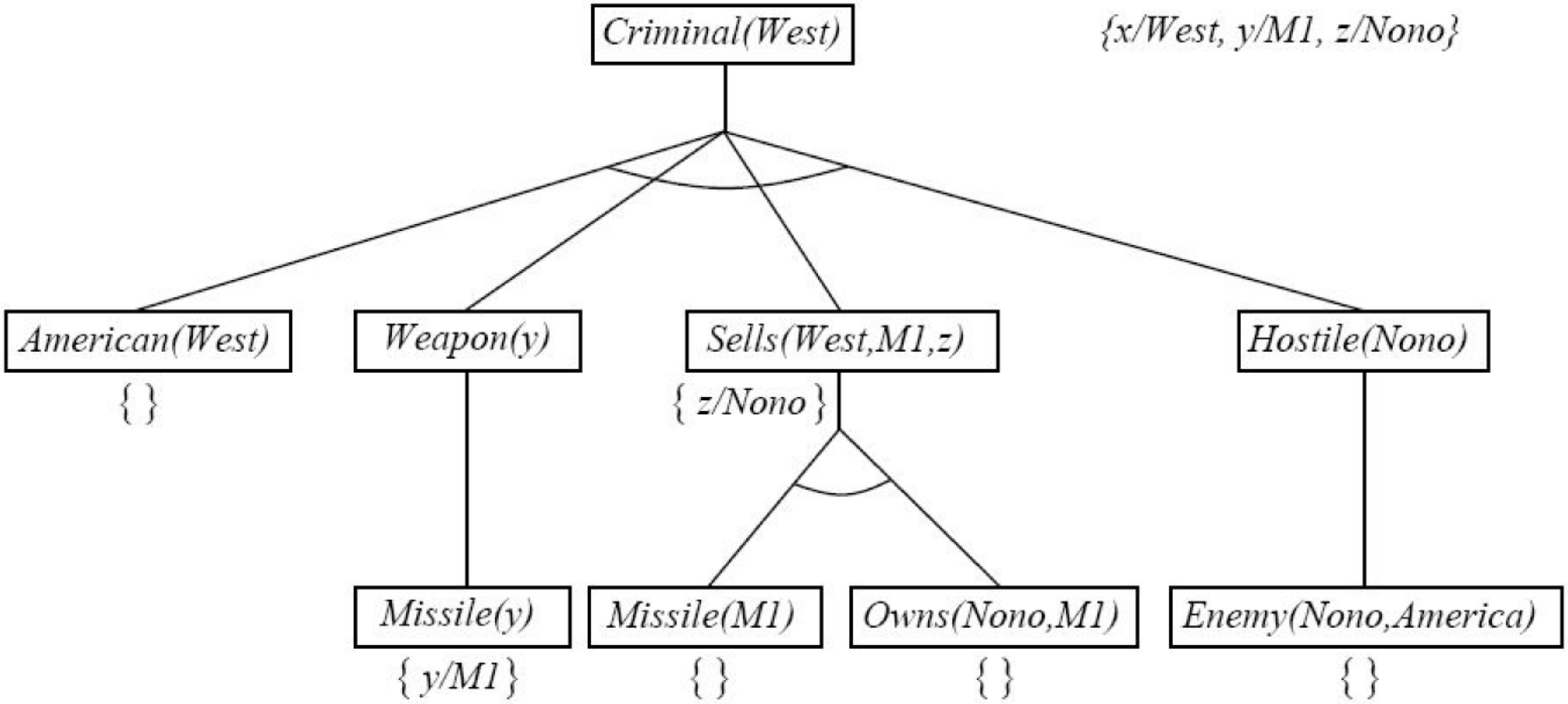



Criminal(West)
{x/West}
American(West)
{ }
Weapon(y)
Sells(x,y,z)
Hostile(z)


Criminal(West)
{x/West}
American(West)
{ }
Weapon(y)
Sells(x,y,z)
Hostile(z)
Missile(y)


Criminal(West)
{x/West, y/M1}
American(West)
{ }
Weapon(y)
Sells(x,y,z)
Hostile(z)
Missile(y)
{ y/M1 }


Criminal(West)
{x/West, y/M1, z/Nono}
American(West)
{ }
Weapon(y)
Sells(West,M1,z)
{ z/Nono }
Hostile(z)
Missile(y)
{ y/M1 }
Missile(M1)
Owns(Nono,M1)


Criminal(West)
{x/West, y/M1, z/Nono}
American(West)
{ }
Weapon(y)
Sells(West,M1,z)
{ z/Nono }
Hostile(Nono)
Missile(y)
{ y/M1 }
Missile(M1)
{ }
Owns(Nono,M1)
{ }
Enemy(Nono,America)
{ }