Artificial Intelligence

Knowledge Representation and Reasoning

MODULE 4

Contents

- **Logical Agents**
- **Knowledge-based agents**
- **Logic in general - models and entailment**
- **Propositional (Boolean) logic**
- Propositional Theorem proving,
- Agents based on Propositional Logic.
- First Order Predicate Logic – Syntax and Semantics of First Order Logic, Using First Order Logic, Knowledge representation in First Order Logic. Inference in First Order Logic – Propositional Vs First Order inference,
- Unification and Lifting,
- Forward chaining,
- Backward chaining,
- Resolution.

Logical Agents

Humans can know “things” and “reason”

Representation: How are things stored?

Reasoning: How is the knowledge used?

- To solve a problem…
- To generate more knowledge…

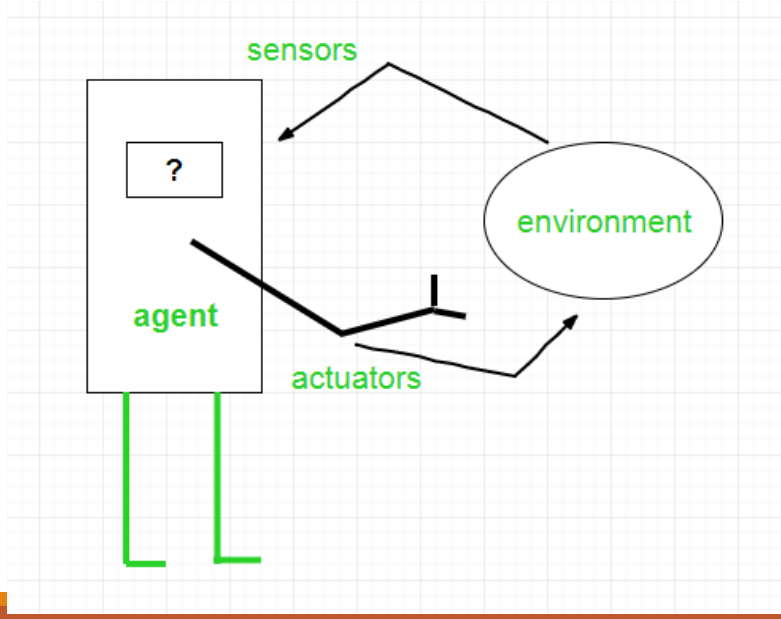
- Knowledge and reasoning are important to artificial agents because they enable successful behaviors difficult to achieve otherwise

– Useful in partially observable environments

Logical Agents

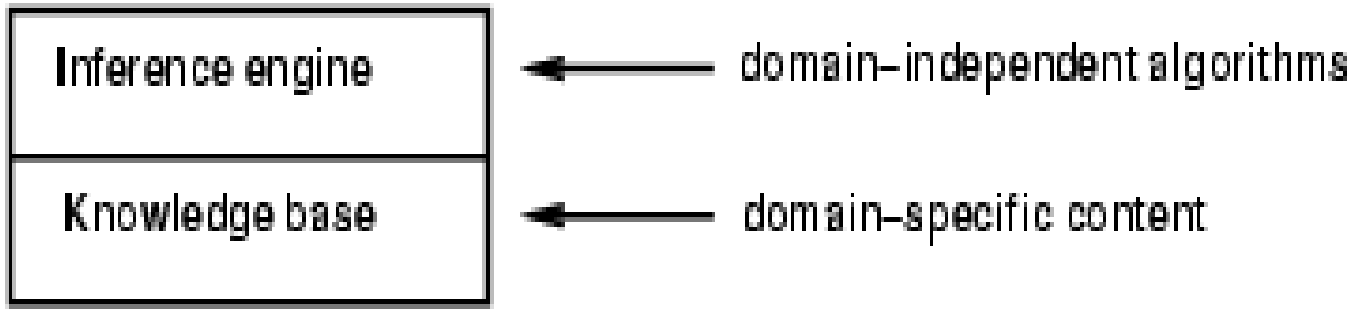
"Logical AI:

The idea is that an agent can represent knowledge of its world, its goals and the current situation by ***sentences in logic*** and decide what to do by inferring that a certain action or course of ***action*** is ***appropriate to achieve its goals***."

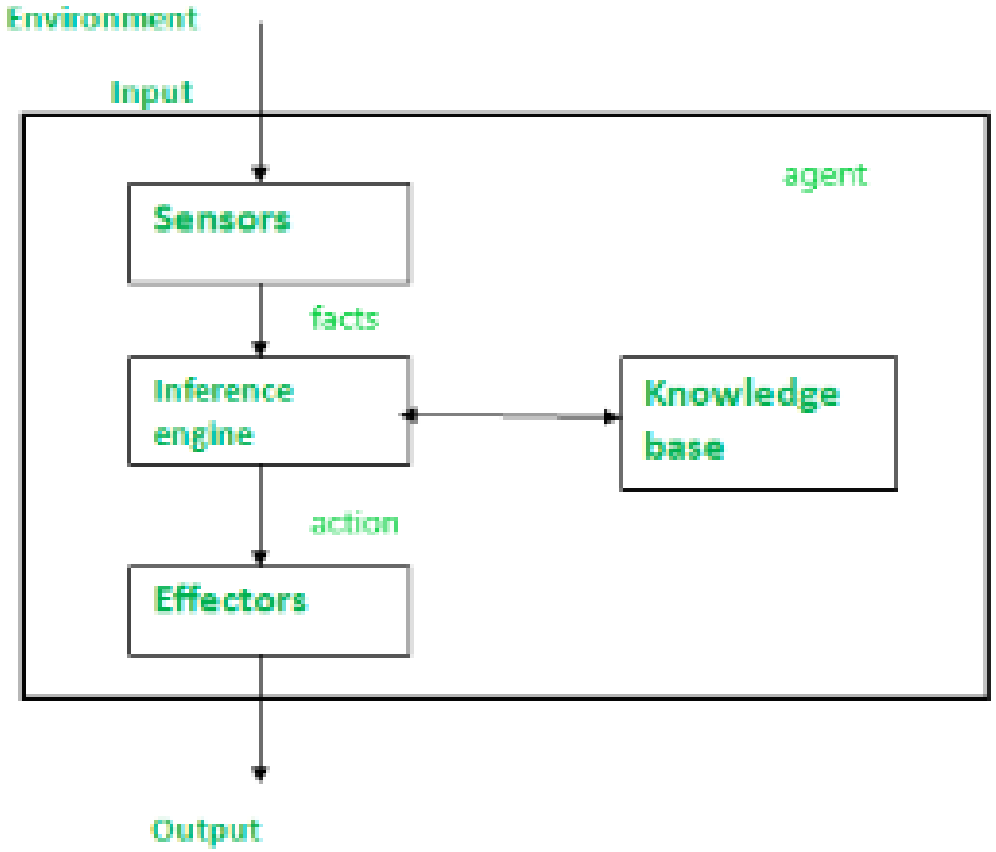


Knowledge-Based Agents

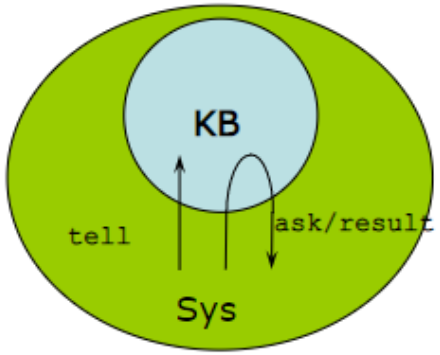
- Central component of a Knowledge-Based Agent is a ***Knowledge-Base***
- A set of sentences in a formal language
- Sentences are expressed using a knowledge representation language
- Two generic functions:
  - **TELL** - add new sentences (facts) to the KB • "Tell it what it needs to know"
  - **ASK** - query what is known from the KB • "Ask what to do next"



Knowledge Base



Knowledge Based Agents



Describing a KB...

What it knows ⇨	Knowledge level
How it knows ⇨	Logical level
Knowledge implementation ⇨	Implementation level

Knowledge Base

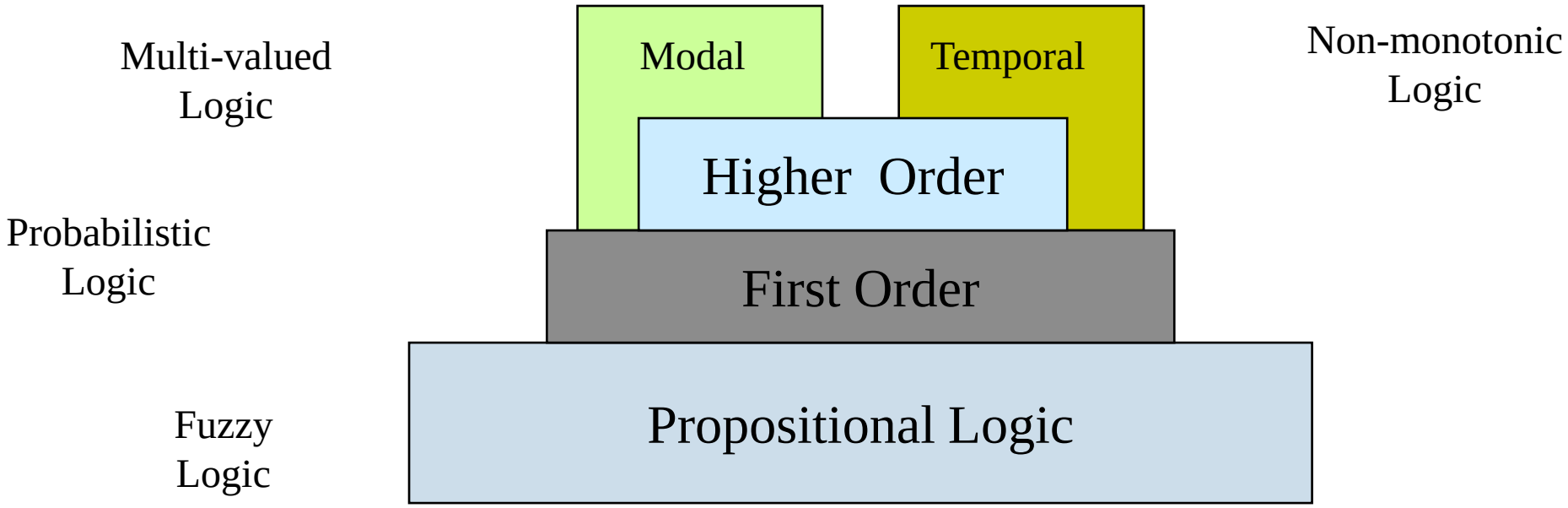
- **Performance Measure:** Performance measure is the unit to define the success of an agent. Performance varies with agents based on their different precepts.
- **Environment**: Environment is the surrounding of an agent at every instant. It keeps changing with time if the agent is set in motion.
- **Actuator**: An actuator is a part of the agent that delivers the output of action to the environment.
- **Sensor**: Sensors are the receptive parts of an agent that takes in the input for the agent.

A simple knowledge-based agent

```
function KB-AGENT(percept) returns an action
    static: KB, a knowledge base
            t, a counter, initially 0, indicating time

    TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
    action ← ASK(KB, MAKE-ACTION-QUERY(t))
    TELL(KB, MAKE-ACTION-SENTENCE(action, t))
    t ← t + 1
    return action
```

Logic as a KR language



Logics

- Logics are formal languages for representing information such that conclusions can be drawn
- Syntax defines the sentences in the language
- Semantics define the "meaning" of sentences; define truth of a sentence in a world

E.g., the language of arithmetic

- $x+2 \geq y$ is a sentence
- $x2+y > \{\}$ is not a sentence
- $x+2 \geq y$ is true iff the number $x+2$ is no less than the number y
- $x+2 \geq y$ is true in a world where $x = 7, y = 1$

Model- “possible world, mathematical abstractions, each of which simply fixes the truth or falsehood of every relevant sentence

possible worlds- real environments that the agent might or might not be in

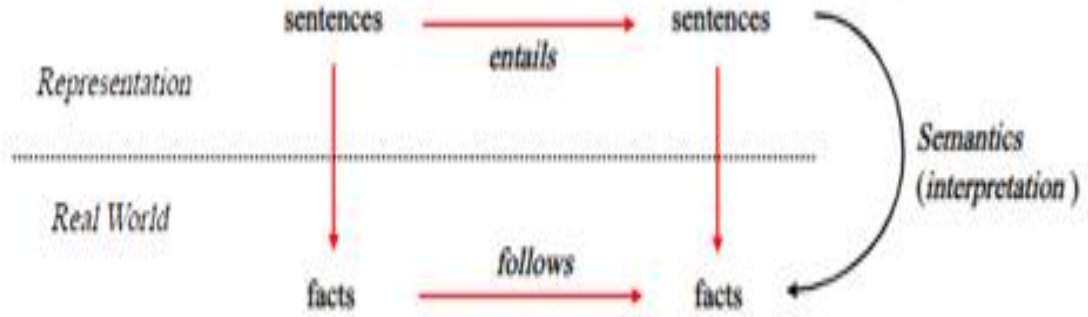
If a sentence α is true in model m, we say that

- m satisfies α or sometimes m is a model of α.

We use the notation M(α) to mean the set of all models of α

Knowledge Representation

- The knowledge representation language provides a declarative representation of real-world objects and their relationships



Entailment

Entailment means a sentence follows logically from another : $\alpha \models \beta$

to mean that the sentence α entails the sentence β

α |= β if and only if, in every model in which α is true, β is also true

α |= β if and only if M(α) ⊆ M(β) .

Knowledge base KB entails sentence α if and only if α is true in all worlds where KB is true

Eg. x+y = 4 entails 4 = x+y

Entailment is a relationship between sentences (i. e. , syntax) that is based on semantics

Inference and Entailment

- Inference is a procedure that allows new sentences to be derived from a knowledge base.

- Understanding inference and entailment: think of

  Set of all consequences of a KB as a haystack
  α as the needle

Entailment is like the needle being in the haystack
Inference is like finding it

if an inference algorithm i can derive α from KB, we write

$$KB \vdash_i \alpha \ ,$$

which is pronounced “α is derived from KB by i” or “i derives α from KB.”

Inference and Entailment

- M is a model of a sentence α if α is true in M
- M(α) is the set of all models of α

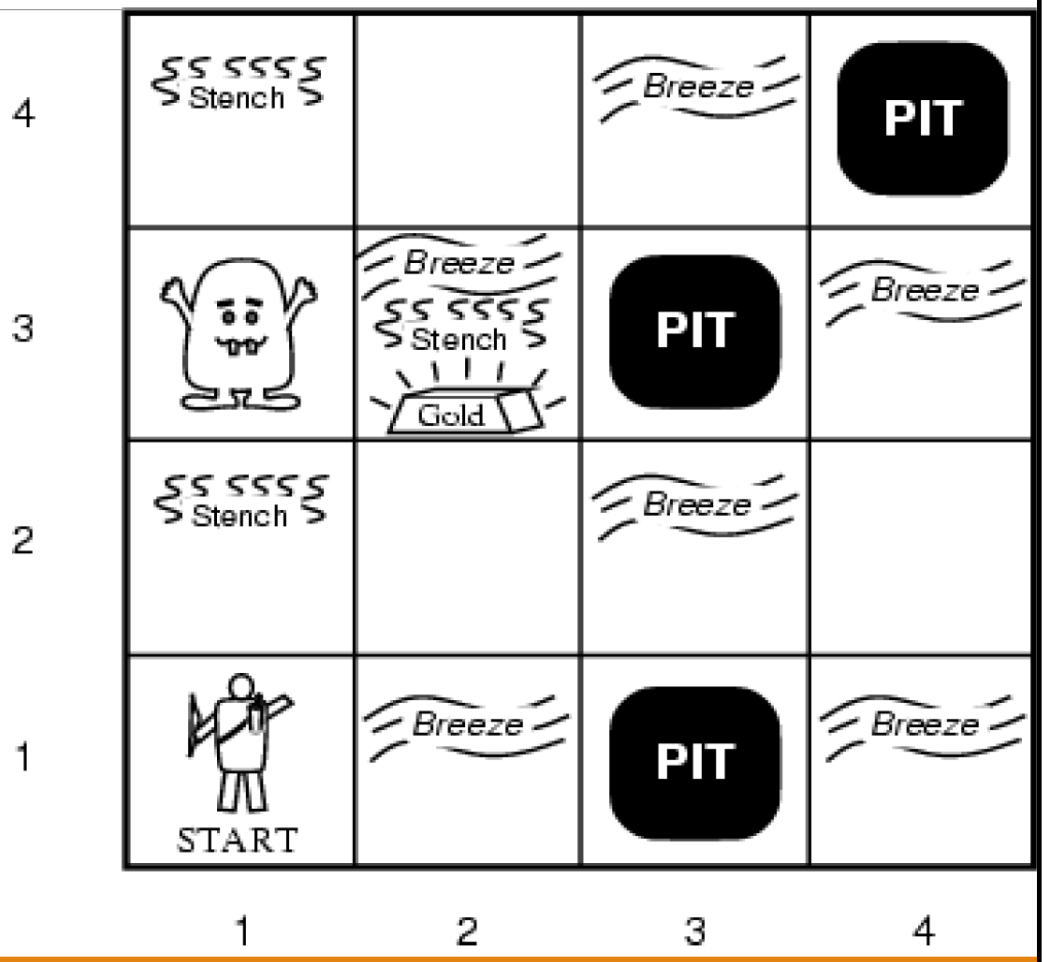
- **Entailment**
  - KB entails a sentence s: $KB \models s$
  - KB derives (proves) a sentence s: $KB \vdash s$
- **Soundness and Completeness**
  - Soundness: $KB \vdash s \Rightarrow KB \models s$, for all s
  - Completeness : $KB \models s \Rightarrow KB \vdash s$, for all s

Validity: true under all interpretations

Satisfiability: true under some interpretation, i.e., there is at least one model

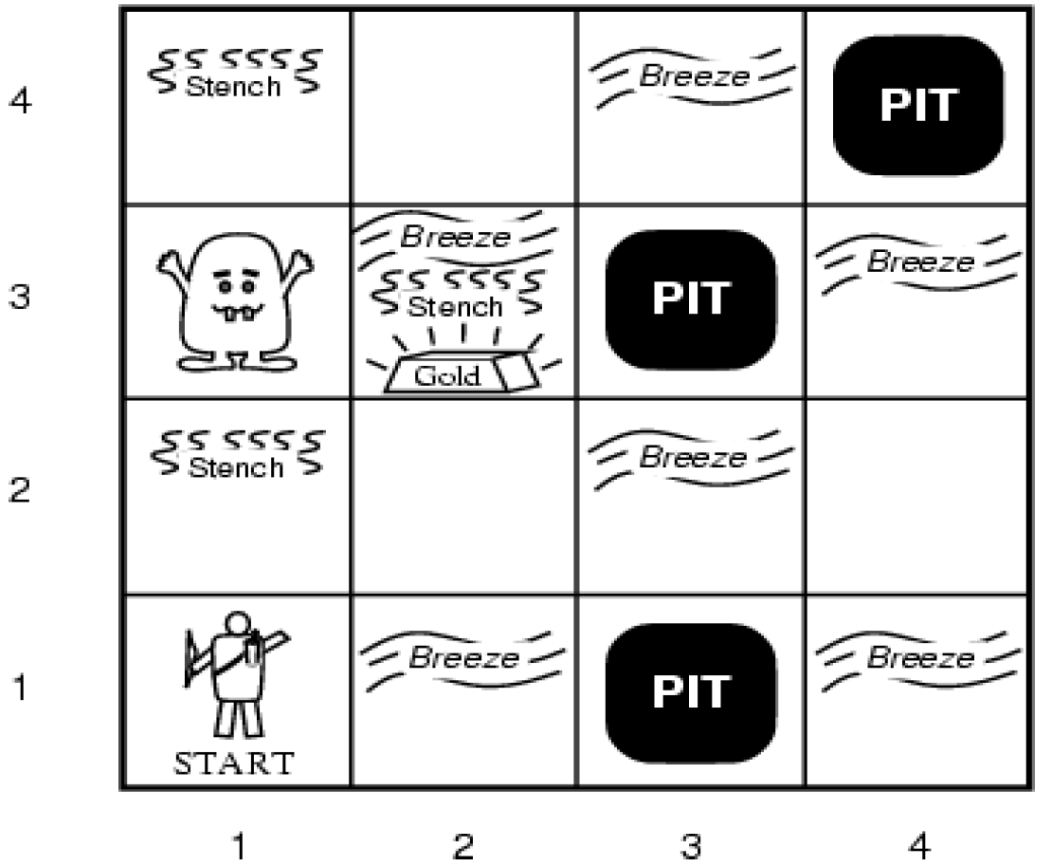
The Wumpus World environment
The Wumpus computer game

- The agent explores a cave consisting of rooms connected by passageways
- Lurking somewhere in the cave is the Wumpus, a beast that eats any agent that enters its room
- Some rooms contain bottomless pits that trap any agent that wanders into the room
- The Wumpus can fall into a pit too, so avoids them
- Occasionally, there is a heap of gold in a room.
- The goal is to collect the gold and exit the world without being eaten



AIMA's Wumpus World

- The agent always starts in the field [1,1]
- Agent's task is to find the gold, return to the field [1,1] and climb out of the cave



Propositional Logic simplest logic

- Syntax of PL: defines the allowable sentences or propositions.

- **Definition (Proposition):** A proposition is a declarative statement (True or False).
  A fact, like "the Sun is hot." The Sun cannot be both hot and not hot at the same time. This declarative statement could also be referred to as a **proposition**.

- **Atomic proposition:** single proposition symbol. Each symbol is a proposition.
  Notation: upper case letters and may contain subscripts. **Eg:-** W_{13}

- **Compound proposition:** constructed from atomic propositions using parentheses and logical connectives.

Atomic Proposition

Examples of atomic propositions:

- 2+2=4 is a true proposition
- $W_{1,3}$ is a proposition. It is true if there is a Wumpus in [1,3]
- "If there is a stench in [1,2] then there is a Wumpus in [1,3]" is a proposition
- "How are you?" or "Hello!" are not propositions. In general, statement that are questions, commands, or opinions are not propositions.

Examples of compound/complex propositions

Let p, p1, and p2 be propositions

- Negation ¬p is also a proposition. A literal is either an atomic proposition or its negation.

  E.g., $W_{1,3}$ is a positive literal, and $\neg W_{1,3}$ is a negative literal.

- Conjunction p1 ∧ p2. E.g., $W_{1,3} \wedge P_{3,1}$

- Disjunction p1 ∨ p2 E.g., $W_{1,3} \vee P_{3,1}$

- Implication p1 → p2. E.g., $W_{1,3} \wedge P_{3,1} \rightarrow \neg W_{2,2}$ (⇒)

- If and only if p1 ↔ p2. E.g., $W_{1,3} \leftrightarrow \neg W_{2,2}$ (⇔)

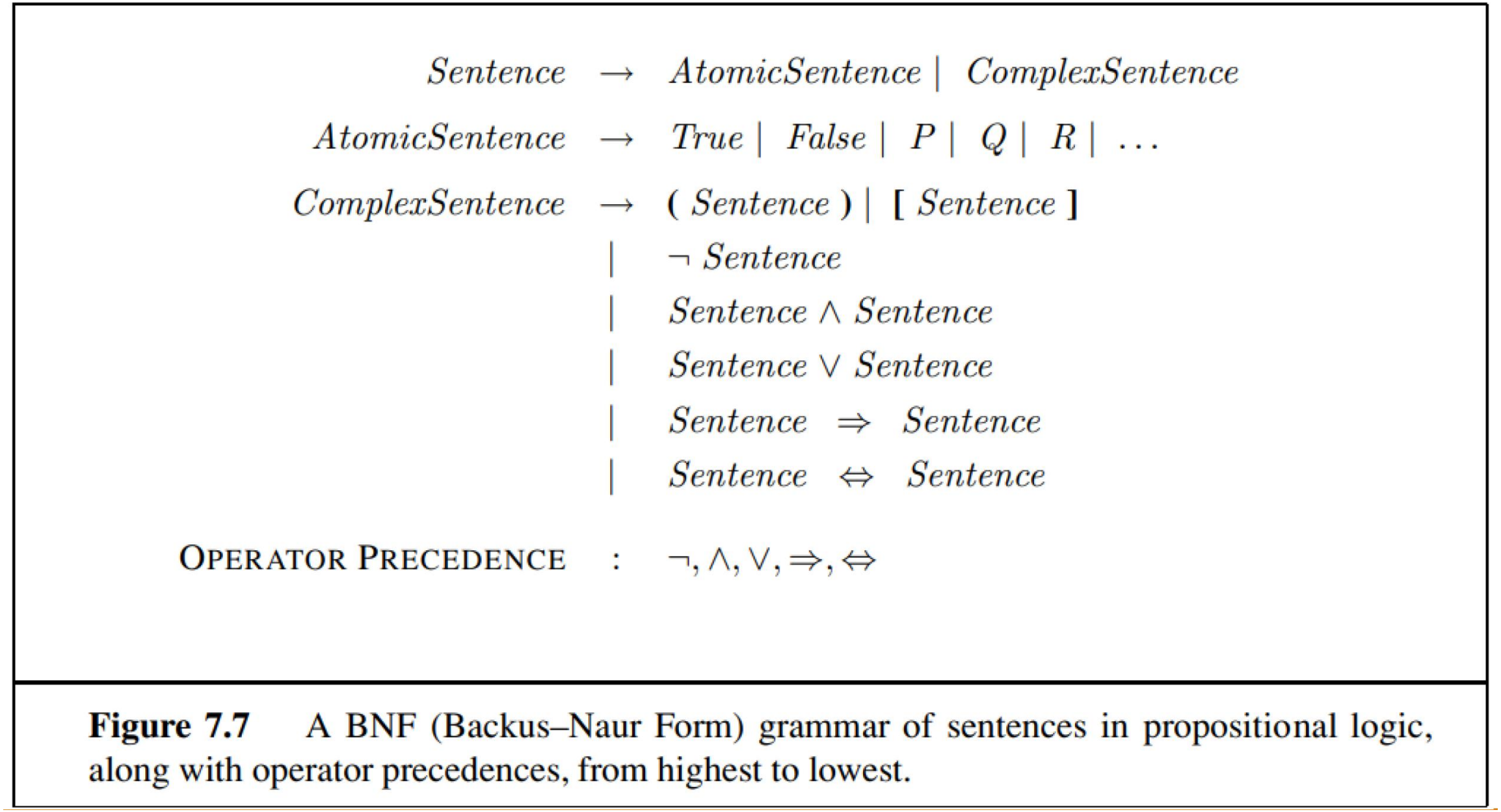


Figure 7.7 A BNF (Backus–Naur Form) grammar of sentences in propositional logic, along with operator precedences, from highest to lowest.

Truth Table

The semantics define the rules to determine the truth of a sentence.

- Semantics can be specified by truth tables.

- Boolean values domain: True ,False

  n-tuple: (x1, x2, ..., xn)

  Operator on n-tuples : g(x1 = v1, x2 = v2, ..., xn = vn)

  Eg :- $m_1 = \{P_{1,2} = \text{false}, P_{2,2} = \text{false}, P_{3,1} = \text{true}\}$

- A truth table defines an operator g on n- tuples by specifying a boolean value for each tuple.

- Number of rows in a truth table? $R = 2^n$

Building Propositions

Negation

p	$\neg p$
T	F
F	T

Conjunction

p	q	$p \wedge q$
T	T	T
T	F	F
F	T	F
F	F	F

If and only if

p	q	$p \leftrightarrow q$
T	T	T
T	F	F
F	T	F
F	F	T

Exclusive OR

p	q	$p \oplus q$
T	T	F
T	F	T
F	T	T
F	F	F

Disjunction

p	q	$p \vee q$
T	T	T
T	F	T
F	T	T
F	F	F

Implication

p	q	$p \rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

Precedence of operators

1. Expressions in parentheses are processed (inside to outside)
2. Negation
3. AND
4. OR
5. Implication
6. Biconditional
7. Left to right

- Use parentheses whenever you have any doubt!

Building proposition

p	q	r	¬r	p ∨ q	p ∨ q → ¬r
T	T	T	F	T	F
T	T	F	T	T	T
T	F	T	F	T	F
T	F	F	T	T	T
F	T	T	F	T	F
F	T	F	T	T	T
F	F	T	F	F	T
F	F	F	T	F	T

Logical Equivalence

Two propositions p and q are logically equivalent if and only if the columns in the truth table giving their truth values agree.

- We write this as p ⇔ q or p ≡ q.

p	q	$\neg p$	$\neg p \vee q$	$p \rightarrow q$
T	T	F	T	T
T	F	F	F	F
F	T	T	T	T
F	F	T	T	T

Properties

- Commutativity:

  $p \wedge q = q \wedge p$

  $p \vee q = q \vee p$

- Associativity:

  $p \wedge q) \wedge r = p \wedge (q \wedge r)$

  $(p \vee q) \vee r = p \vee (q \vee r)$

- Identity element:

  $p \wedge True = p$

  $p \vee True = True$

- $\neg(\neg p) = p$
- $p \wedge p = p$ $p \vee p = p$

- Distributivity:

  $p \wedge (q \vee r) = (p \wedge q) \vee (p \wedge r)$

  $p \vee (q \wedge r) = (p \vee q) \wedge (p \vee r)$

- $p \wedge (\neg p) = False$ and $p \vee (\neg p) = True$

- DeMorgan's laws:

  $\neg(p \wedge q) = (\neg p) \vee (\neg q)$

  $\neg(p \vee q) = (\neg p) \wedge (\neg q)$

Technical Terms: Soundness and Completeness

Soundness: inference proves only true things.

Completeness: all true things can be proved.

More formally,
KB $|=_i \alpha$ means that sentence α can be derived from KB by a procedure i.

***Soundness*:** i is sound if:
whenever KB $|=_i \alpha$, it is also true that KB $|= \alpha$.

***Completeness*:** i is complete if:
whenever KB $|= \alpha$, it is also true that KB $|=_i \alpha$.

Tautology and contradiction

- Tautology is a proposition which is always true
- Contradiction is a proposition which is always false
- Contingency is a proposition which is neither a tautology or a contradiction

P	$\neg p$	$p \vee \neg p$	$p \wedge \neg p$
T	F	T	F
F	T	T	F

Contrapositive and Inverse

Given an implication p → q

- The converse is: q → p
- The contrapositive is: ¬q → ¬p
- The inverse is: ¬p → ¬q

Conditional statement: “If it is raining, then the grass is wet.”

The first step is to identify the hypothesis and conclusion statements. Hypothesis, p: it is raining
Conclusion, q: grass is wet

Converse statement would be: “If the grass is wet, then it is raining.”

Inverse statement would be: “If it is NOT raining, then the grass is NOT wet.”

Contrapositive statement would be: “If the grass is NOT wet, then it is NOT raining.”

Inference (Modus Ponens)

$$\frac{p \qquad p \rightarrow q}{q}$$

- Assume you are given the following two statements:
  - "you are in this class"
  - "if you are in this class, you will get a grade"

$$\frac{p \qquad p \rightarrow q}{\therefore q}$$

- Let p = "you are in this class"
- Let q = "you will get a grade"

- By Modus Ponens, you can conclude that you will get a grade

Modus ponens- Truth Table

- Consider $(p \wedge (p \rightarrow q)) \rightarrow q$

p	q	p→q	p∧(p→q))	(p∧(p→q)) → q
T	T	T	T	T
T	F	F	F	T
F	T	T	F	T
F	F	T	F	T

$$\begin{array}{l} p \\ \underline{p \rightarrow q} \\ \therefore q \end{array}$$

Implication

p	q	$p \rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

Modus Tollens

Modus tollens is a valid argument form in propositional calculus in which p and q are propositions. If p implies q, and q is false, then p is false. Also known as an indirect proof or a proof by contrapositive.

$$\frac{p \Rightarrow q, \neg q}{\therefore \neg p}$$

For example, if being the king implies having a crown, not having a crown implies not being the king.

- Assume that we know: ¬q and p → q
  - Recall that p → q = ¬q → ¬p
- Thus, we know ¬q and ¬q → ¬p
- We can conclude ¬p

$$\frac{\neg q \qquad p \rightarrow q}{\therefore \neg p}$$

- Assume you are given the following two statements:
  - "you will not get a grade"
  - "if you are in this class, you will get a grade"

$$\frac{\neg q \qquad p \rightarrow q}{\therefore \neg p}$$

- Let p = "you are in this class"
- Let q = "you will get a grade"

- By Modus Tollens, you can conclude that you are not in this class

And Elimination, Unit Resolution

And-Elimination : (From a conjunction, you can infer any of the conjuncts.)

$$\frac{\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n}{\alpha_i}$$

from a conjunction, any of the conjuncts can be inferred.
For example, from (WumpusAhead ∧ WumpusAlive), WumpusAlive can be inferred.

Unit Resolution : (From a disjunction, if one of the disjuncts is false, then you can infer the other one is true.)

$$\frac{\alpha \vee \beta, \ \neg \beta}{\alpha}$$

DNF and CNF

A normal form is a canonical way to write a logical sentence. It turns out that any sentence can be written in either of the *universal* forms:

DNF: Disjunctive Normal Form

OR of ANDs (terms)

e.g. $(p \wedge \neg q) \vee (\neg p \wedge \neg r)$

CNF: Conjunctive Normal Form

"every sentence of propositional logic is logically equivalent to a conjunction of clauses"

AND of ORs (clauses)

e.g. $(p \vee \neg q) \wedge (\neg p \vee \neg r)$

procedure for converting to CNF

Convert the sentence B1,1 ⇔ (P1,2 ∨ P2,1) into CNF

1. **Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.**

$$(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1}) .$$

2. **Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$:**

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1}) .$$

3. CNF requires $\neg$ to appear only in literals, so we "move $\neg$ inwards" by repeated application of the following equivalences from Figure 7.11:

$$\neg(\neg\alpha) \equiv \alpha \quad \text{(double-negation elimination)}$$
$$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta) \quad \text{(De Morgan)}$$
$$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta) \quad \text{(De Morgan)}$$

In the example, we require just one application of the last rule:

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1}) .$$

4. Now we have a sentence containing nested $\wedge$ and $\vee$ operators applied to literals. We apply the distributivity law from Figure 7.11, distributing $\vee$ over $\wedge$ wherever possible.

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1}) .$$

Rules to convert PL to CNF

1. Remove Bicondition using rule

$$\alpha \leftrightarrow \beta \equiv (\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$$

2. Remove implication using rule

$$\alpha \rightarrow \beta \equiv \neg\alpha \vee \beta$$

3. Move negation inwards.

$$\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$$

$$\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$$

$$\neg(\neg\alpha) \equiv \alpha$$

4. Apply Distributive and/or Commutative law

$$\alpha \wedge (\beta \vee \gamma) \equiv (\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$$
$$\alpha \vee (\beta \wedge \gamma) \equiv (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$$

} Distributive

$$\alpha \wedge \beta \equiv \beta \wedge \alpha$$
$$\alpha \vee \beta \equiv \beta \vee \alpha$$

} commutative law.

Resolution

Theorem proving by means of contradiction

Many statements and we need conclusions

A resolution algorithm

function PL-RESOLUTION(KB, α) **returns** *true* or *false*
  inputs: KB, the knowledge base, a sentence in propositional logic
    α, the query, a sentence in propositional logic

  clauses $\leftarrow$ the set of clauses in the CNF representation of $KB \wedge \neg\alpha$
  new $\leftarrow \{\}$
  loop do
    for each pair of clauses C_i, C_j **in** *clauses* **do**
      resolvents $\leftarrow$ PL-RESOLVE(C_i, C_j)
      if *resolvents* contains the empty clause **then return** *true*
      new $\leftarrow$ *new* $\cup$ *resolvents*
    if *new* $\subseteq$ *clauses* **then return** *false*
    clauses $\leftarrow$ *clauses* $\cup$ *new*

Figure 7.12 A simple resolution algorithm for propositional logic. The function PL-RESOLVE returns the set of all possible clauses obtained by resolving its two inputs.

First, (KB ∧ ¬α) is converted into CNF.

Then, the resolution rule is applied to the resulting clauses.

Each pair that contains complementary literals is resolved to produce a new clause, which is added to the set if it is not already present.

The process continues until one of two things happens:

- there are no new clauses that can be added, in which case KB does not entail α; or,
- two clauses resolve to yield the empty clause, in which case KB entails α.

The empty clause—a disjunction of no disjuncts—is equivalent to False because a disjunction is true only if at least one of its disjuncts is true.

Another way to see that an empty clause represents a contradiction is to observe that it arises only from resolving two complementary unit clauses such as P and ¬P.

- Tom is hardworking

P: Hardworking(Tom)

- Tom is an intelligent student

Q:Intelligent(Tom)

- If Tom is hardworking and intelligent then Tom scores high marks

R: Hardworking(Tom) ∧Intelligent(Tom) →ScoresHighMarks(Tom)

What about other students that are hardworking and intelligent?

All students that are hardworking and intelligent score high marks

Let all students be represented by variable ‘x’

$$Student(x) \land Hardworking(x) \land Intelligent(x) \rightarrow ScoresHighMarks(x)$$

Demerit of Propositional Logic

Propositional logic can only represent the facts, which are either true or false.

PL is not sufficient to represent the complex sentences or natural language statements.

The propositional logic has very limited expressive power.

Consider the following sentence, which we cannot represent using PL logic.

- **"Some humans are intelligent", or**
- **"Sachin likes cricket."**

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.

First-order Logic

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, bigger-than, outside..

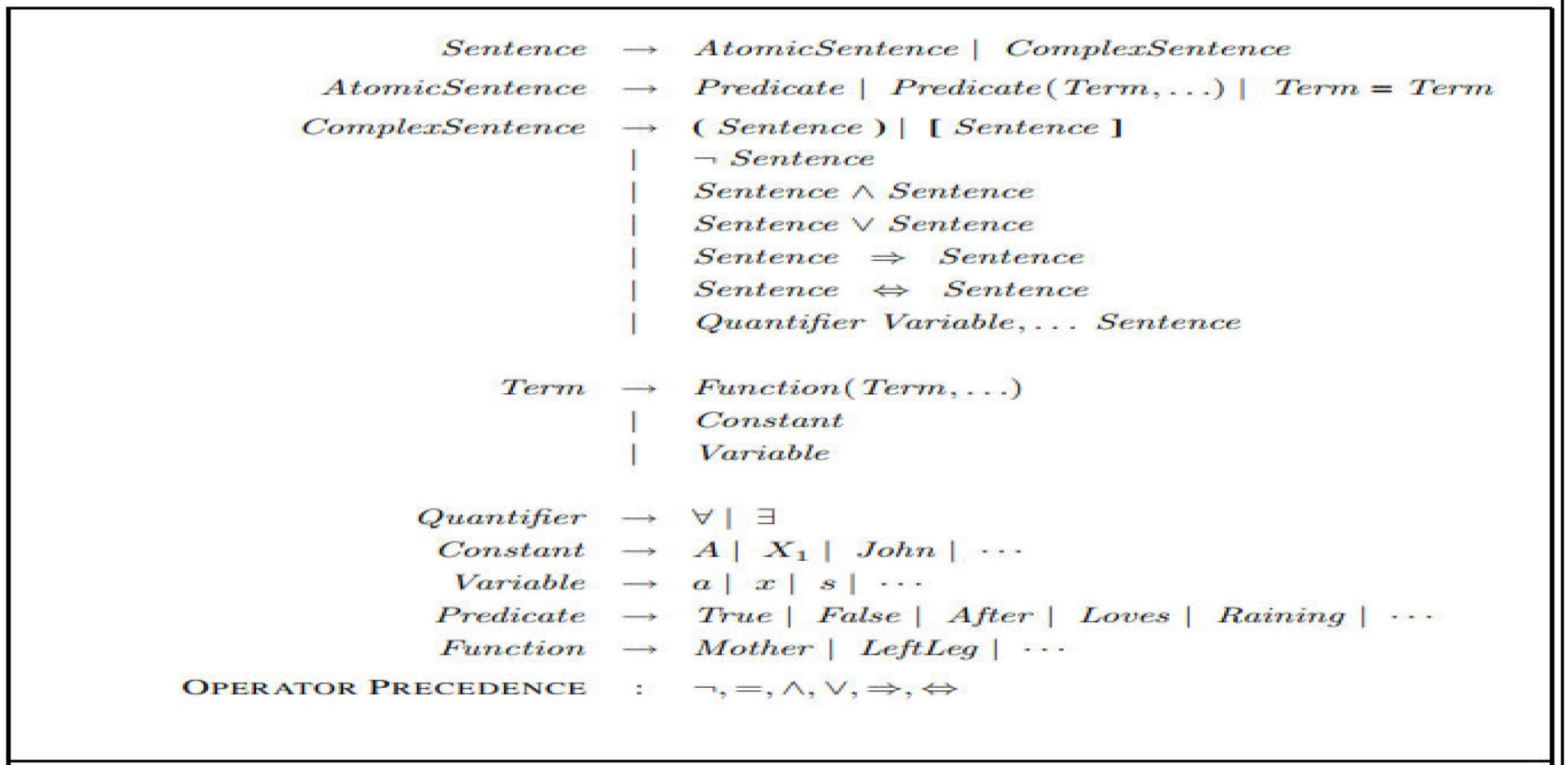
Properties: blue, oval, even, large, ...

Functions: father-of, best-friend, second-half, one-more-than ...

Also called as Predicate Logic

It is a generalization of Propositional Logic that allows us to express and infer arguments in infinite models, Eg,

- Some birds can fly
- All men are mortal
- At least one student has course registered



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

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.

Basic Element in FOL

Constant	1, 2, A, John, Mumbai, cat,....
Variables	x, y, z, a, b,....
Predicates	Brother, Father, >,....
Function	sqrt, LeftLegOf, ...
Connectives	∧, ∨, ¬, ⇒, ⇔
Equality	==
Quantifier	∀, ∃

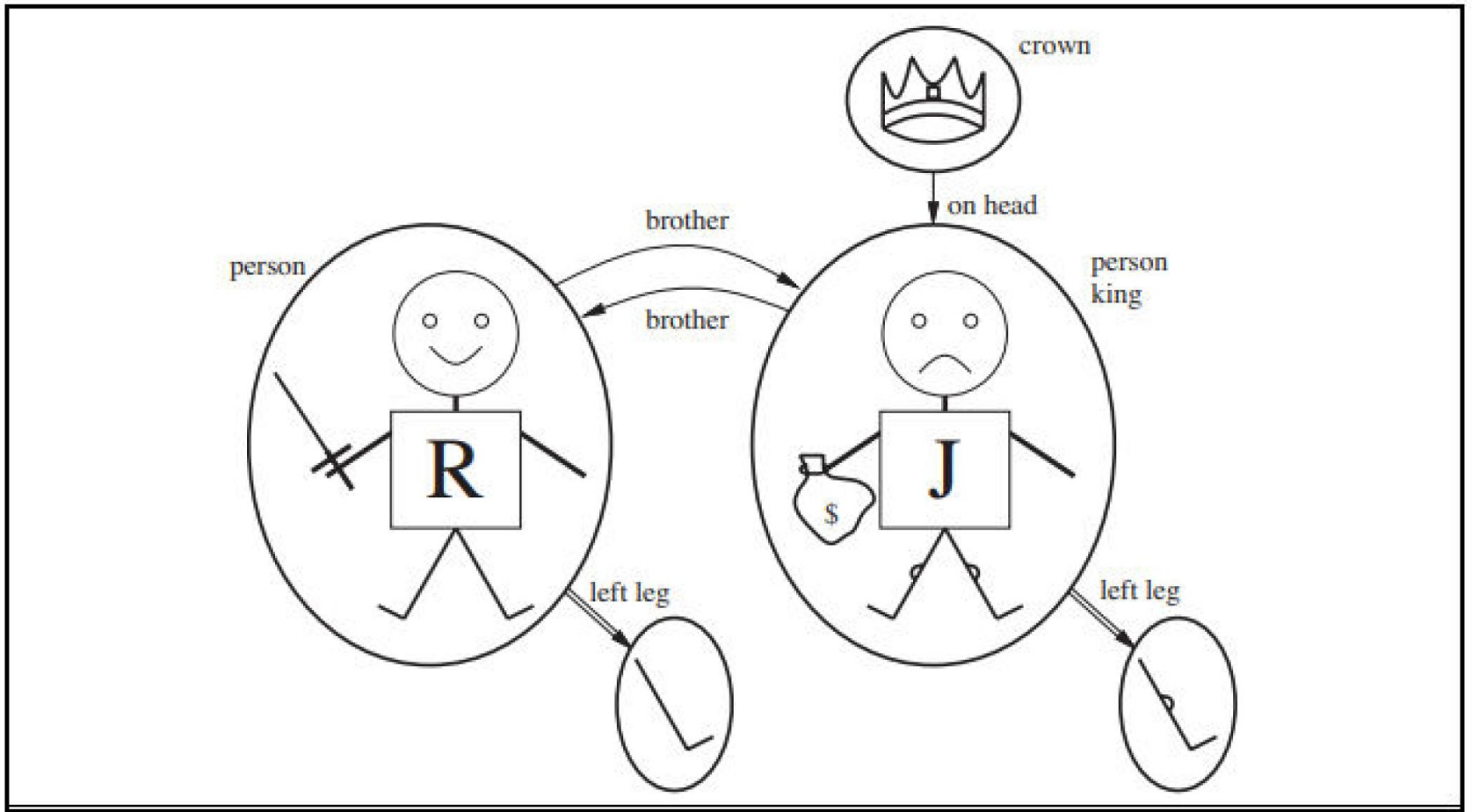


Figure 8.2 A model containing five objects, two binary relations, three unary relations (indicated by labels on the objects), and one unary function, left-leg.

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.

For example, in English we might use the expression “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).

Consider a term f(t1,...,tn). The function symbol f refers to some function in the model (call it F); the argument terms refer to objects in the domain (call them d1,...,dn); and the term as a whole refers to the object that is the value of the function F applied to d1,...,dn.

Atomic Sentences in FOL

- Atomic sentences are the most basic sentences of first-order logic. These sentences are formed from a predicate symbol followed by a parenthesis with a sequence of terms.
- Atomic sentences are represented as **Predicate (term1, term2, ……, term n)**.

Example: Ravi and Ajay are brothers: => Brothers(Ravi, Ajay).
Chinky is a cat: => cat (Chinky).

Complex sentences

Complex sentences are made by combining atomic sentences using connectives.

We can use logical connectives to construct more complex sentences, with the same syntax and semantics as in propositional calculus.

Here are four sentences that are true in the model of Figure 8.2 under our intended interpretation:

¬Brother (LeftLeg(Richard), John)

Brother (Richard, John) ∧ Brother (John, Richard)

King(Richard) ∨ King(John)

¬King(Richard) ⇒ King(John)

Quantifiers

Quantifiers express properties of entire collections of objects

First-order logic contains two standard quantifiers, called

- universal and
- existential.

Universal quantification (∀)

expression of general rules in propositional logic

The second rule, “All kings are persons,” is written in first-order logic as

∀ x King(x) ⇒ Person(x) // “For all x, if x is a king, then x is a person.”

∀ 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.

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

would be equivalent to asserting

Richard the Lionheart is a king $\wedge$ Richard the Lionheart is a person,
King John is a king $\wedge$ King John is a person,
Richard's left leg is a king $\wedge$ 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) $\wedge$ 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 Sibling(x, y) $\Leftrightarrow$ 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 Loves(x, y) .

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

$\exists$ y $\forall$ x Loves(x, y) .

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:

∀ 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)

De Morgan rules for quantified and unquantified sentences

$$\forall x \ \neg P \equiv \neg \exists x \ P \qquad \neg(P \vee Q) \equiv \neg P \wedge \neg Q$$
$$\neg \forall x \ P \equiv \exists x \ \neg P \qquad \neg(P \wedge Q) \equiv \neg P \vee \neg Q$$
$$\forall x \ P \equiv \neg \exists x \ \neg P \qquad P \wedge Q \equiv \neg(\neg P \vee \neg Q)$$
$$\exists x \ P \equiv \neg \forall x \ \neg P \qquad P \vee Q \equiv \neg(\neg P \wedge \neg Q) \ .$$

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.

Because an interpretation fixes the referent of any term, determining the truth of an equality sentence is simply a matter of seeing that the referents of the two terms are the same object.

To say that Richard has at least two brothers, we would write

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

FOL

Points to remember:

- The main connective for universal quantifier ∀ is implication →.
- The main connective for existential quantifier ∃ is and ∧.

Properties of Quantifiers:

- In universal quantifier, ∀x∀y is similar to ∀y∀x.
- In Existential quantifier, ∃x∃y is similar to ∃y∃x.
- ∃x∀y is not similar to ∀y∃x.

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.

∀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 ∀, and it will be represented as follows:

∀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 **∃, and it will be represented as**:

∃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 **∀ with negation, so** following representation for this:

¬∀ (x) [student(x) → like(x, Mathematics) ∧ 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:

∃(x) [student(x) → failed (x, Mathematics) ∧∀ (y) [¬(x==y) ∧ 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.

Emily is either a surgeon or a lawyer.

Occupation(Emily, Surgeon) ∨ Occupation(Emily, Lawyer)

or

Occupation(Emily, Surgeon) ⇔ ¬Occupation(Emily, Lawyer)

Joe is an actor, but he holds another job.

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

or

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

All surgeons are doctors.

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

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

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

Or

$\neg\ \exists\ p\ [Occupation(p, Lawyer) \wedge Customer(Joe, p)]$

or

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

Emily has a boss who is a lawyer.

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

There exists a lawyer all of whose clients are doctors (i.e., all of whose customers are doctors).

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

Or

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

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: ∀x ∃(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: ∀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.

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 Ψ_1 = King(x), Ψ_2 = King(John),
Substitution θ = {John/x} is a unifier for these atoms and applying this substitution, and both expressions will be identical.

Find the MGU of {p(f(a), g(Y)) and p(X, X)}

Sol: S_0 => Here, Ψ_1 = p(f(a), g(Y)), and Ψ_2 = p(X, X)

SUBST θ= {f(a) / X}

S1 => Ψ_1 = p(f(a), g(Y)), and Ψ_2 = p(f(a), f(a))

SUBST θ= {f(a) / g(y)}, **Unification failed.**

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.

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 logic.

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

b. food(Apple) ∧ food(vegetables)

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

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

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

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

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

h. likes(John, Peanuts)

Step-2: Conversion of FOL into CNF

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

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

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

- **Rename variables or standardize variables**
  a. ∀x ¬ food(x) V likes(John, x)
  b. food(Apple) Λ food(vegetables)
  c. ∀y ∀z ¬ eats(y, z) V killed(y) V food(z)
  d. eats (Anil, Peanuts) Λ alive(Anil)
  e. ∀w¬ eats(Anil, w) V eats(Harry, w)
  f. ∀g ¬killed(g)] V alive(g)
  g. ∀k ¬ alive(k) V ¬ killed(k)
  h. likes(John, Peanuts).

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)

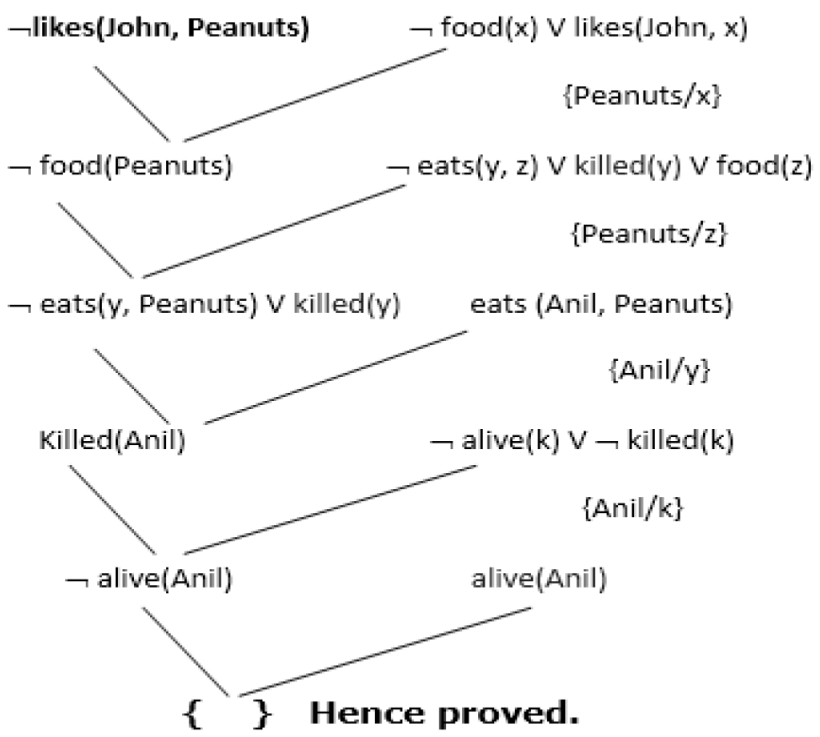
h. killed(g) V alive(g)

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)



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.

Given a new fact, generate all consequences

Assumes all rules are of the form

- C1 and C2 and C3 and…. --> Result

Each rule & binding generates a new fact

This new fact will “trigger” other rules

Keep going until the desired fact is generated

(Semi-decidable as is FOL in general)

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

$American(x) \land Weapon(y) \land Sells(x,y,z) \land Hostile(z) \Rightarrow Criminal(x)$

Nono…has some missiles

$\exists x\ Owns(Nono, x) \land Missiles(x)$

$Owns(Nono, M_1)$ and $Missle(M_1)$

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

$\forall x\ Missle(x) \land Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$

Missiles are weapons

$Missle(x) \Rightarrow Weapon(x)$

An enemy of America counts as “hostile”

Enemy(x, America) ⇒ Hostile(x)

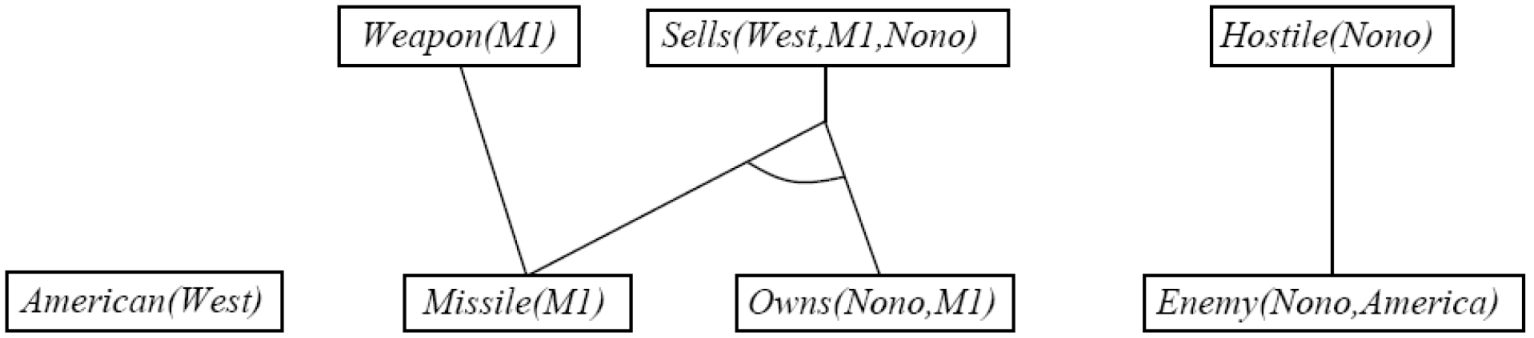
Col. West who is an American

American(Col. West)

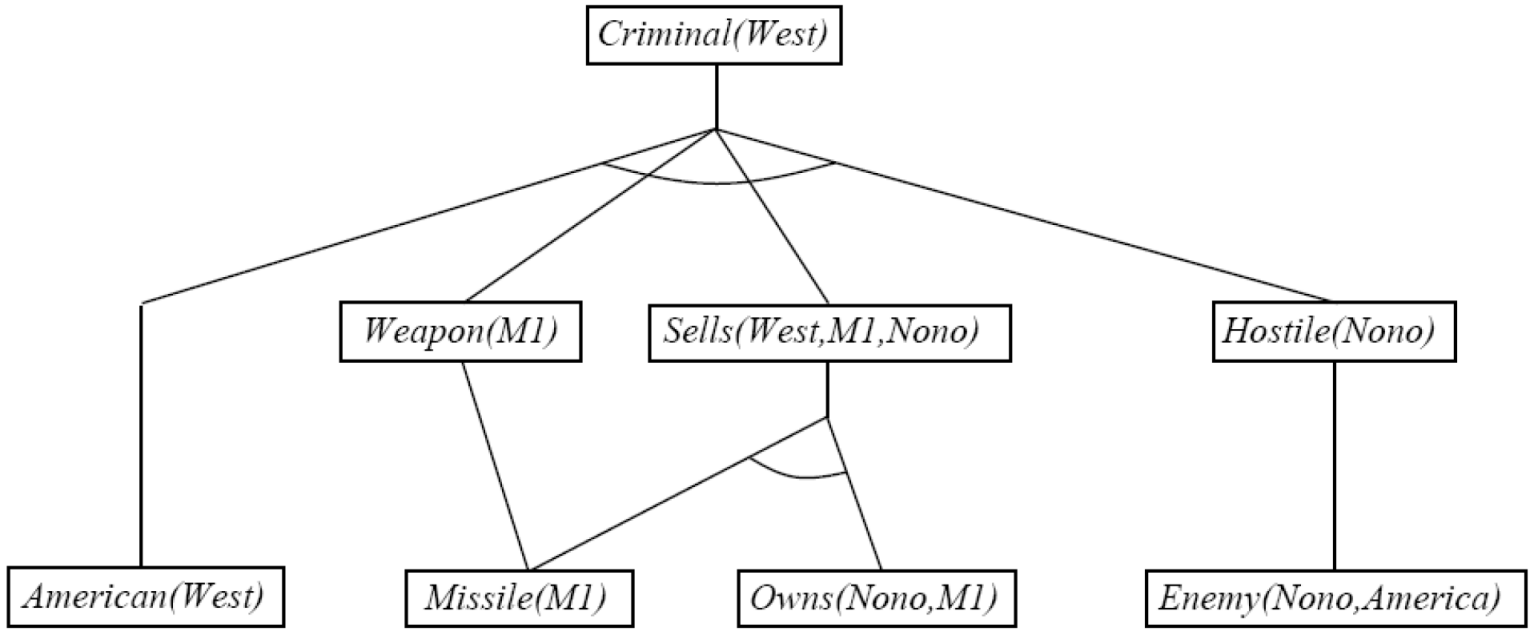
The country Nono, an enemy of America

Enemy(Nono, America)

FC: Example Knowledge Base



FC: Example Knowledge Base



Backward Chaining

Consider the item to be proven a goal

Find a rule whose head is the goal (and bindings)

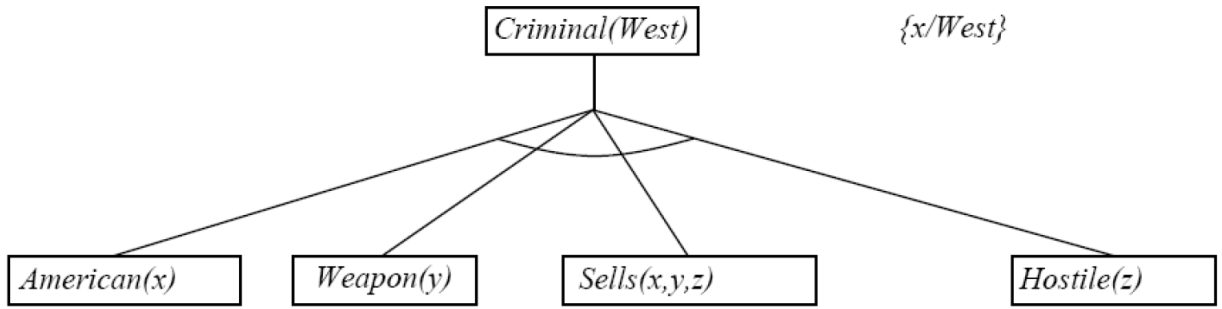
Apply bindings to the body, and prove these (subgoals) in turn

If you prove all the subgoals, increasing the binding set as you go, you will prove the item.

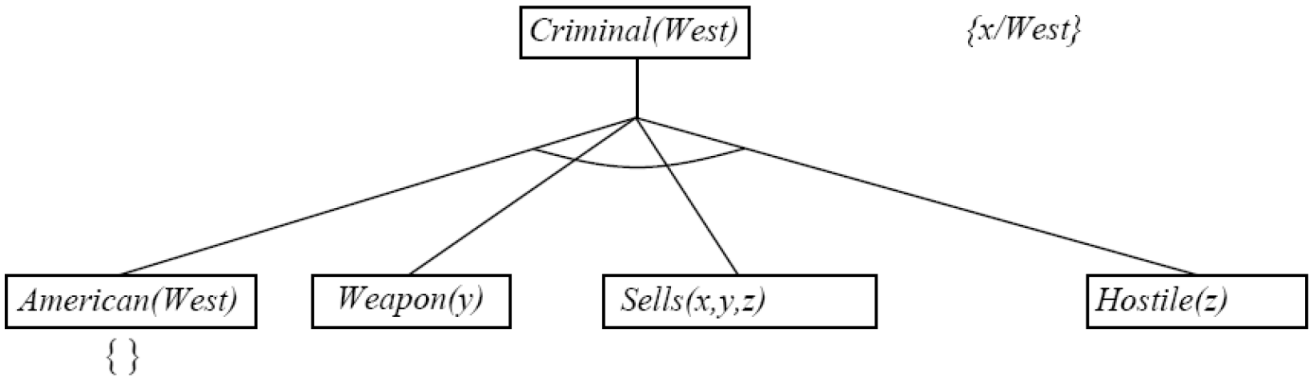
Backward Chaining Example

$Criminal(West)$

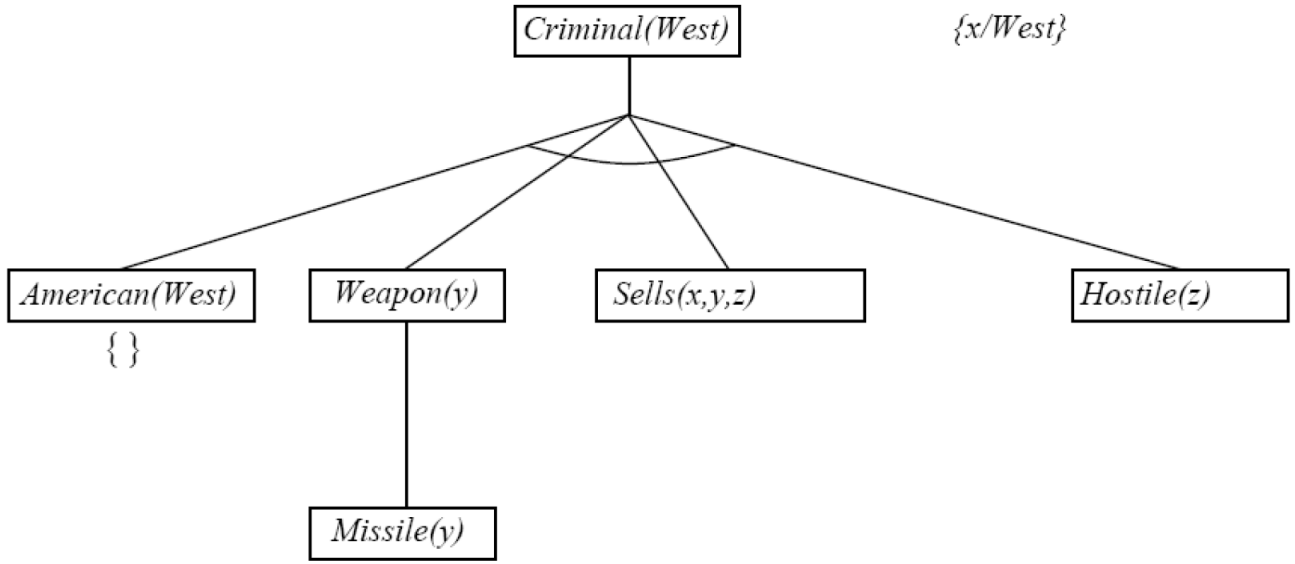
Backward Chaining Example



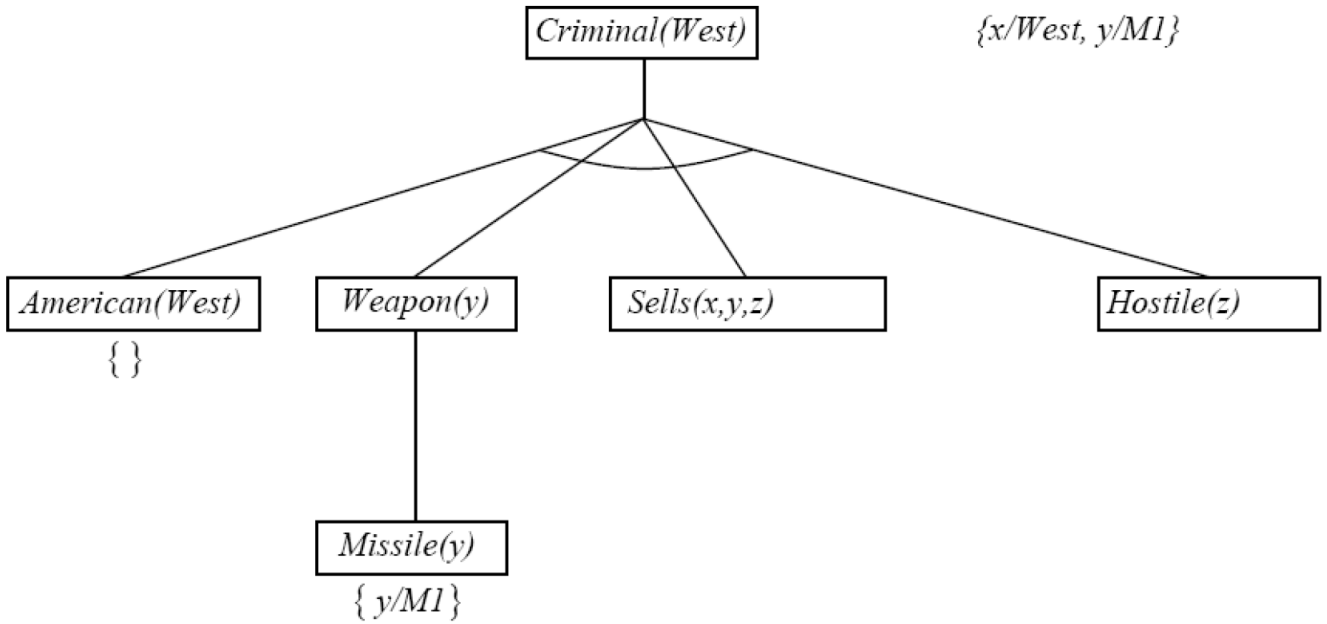
Backward Chaining Example



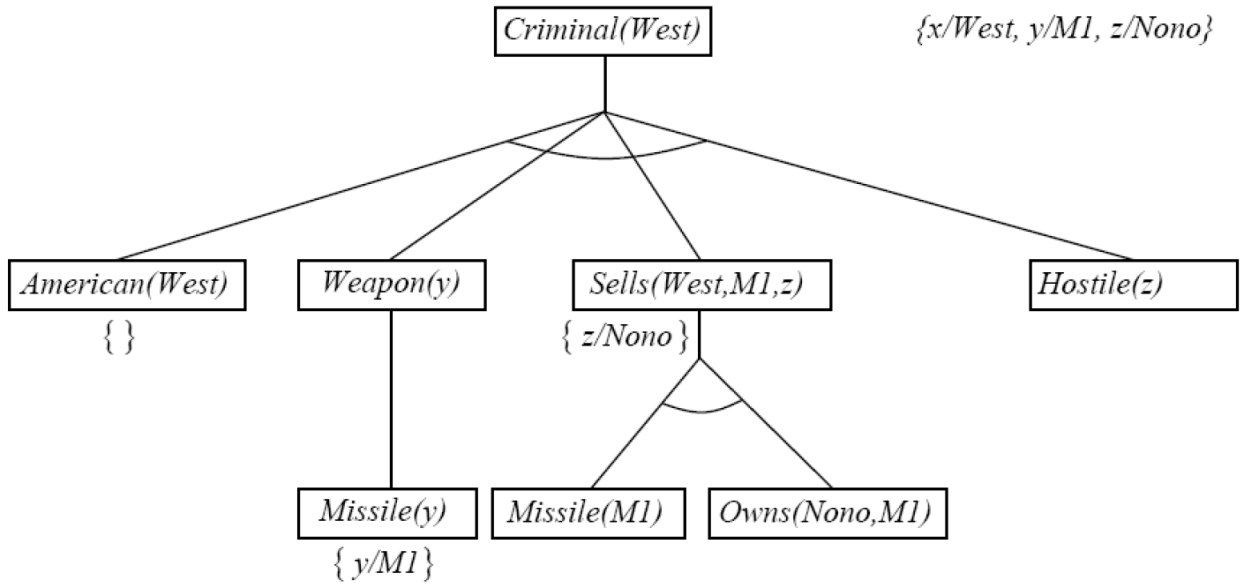
Backward Chaining Example



Backward Chaining Example



Backward Chaining Example



Backward Chaining Example