KNOWLEDGE REPRESENTATION

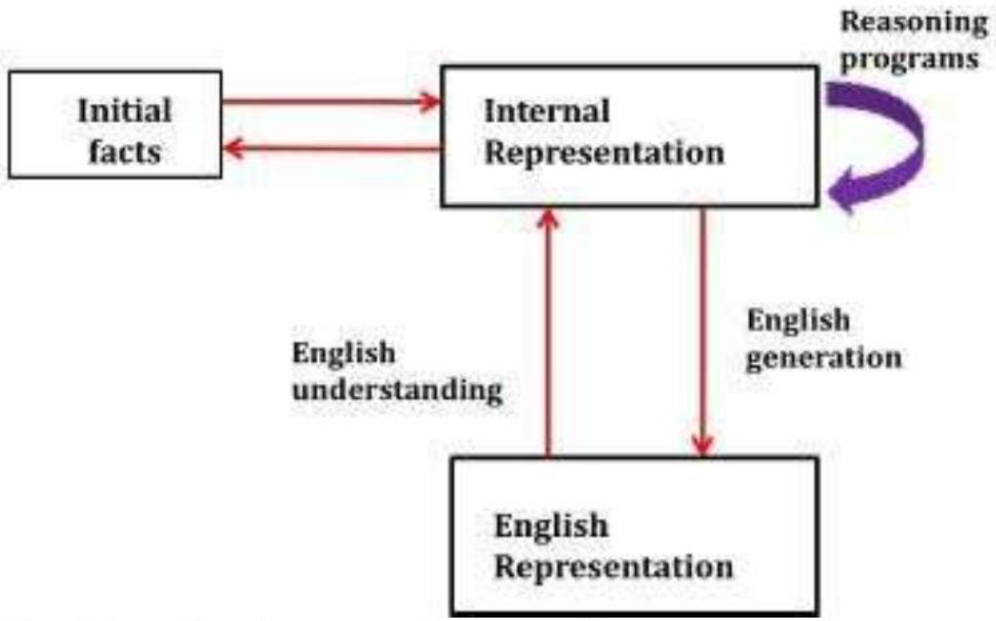


Fig: Mapping between Facts and Representations

- The Knowledge Representation models/mechanisms are often based on:
- Logic
- Rules
- Frames
- Semantic Net

Knowledge is categorized into two major types:

1. Tacit corresponds to “informal” or “implicit“
2. Explicit formal type of knowledge, Explicit

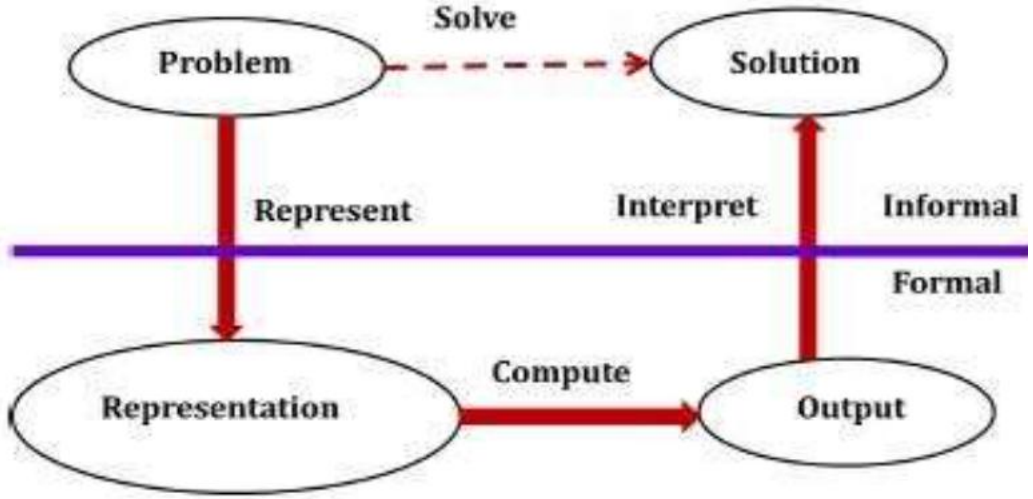


Fig: Knowledge Representation Framework

A knowledge representation system should have following properties.

1. Representational Adequacy
The ability to represent all kinds of knowledge that are needed in that domain.

2. Inferential Adequacy
Also, The ability to manipulate the representational structures to derive new structures corresponding to new knowledge inferred from old.

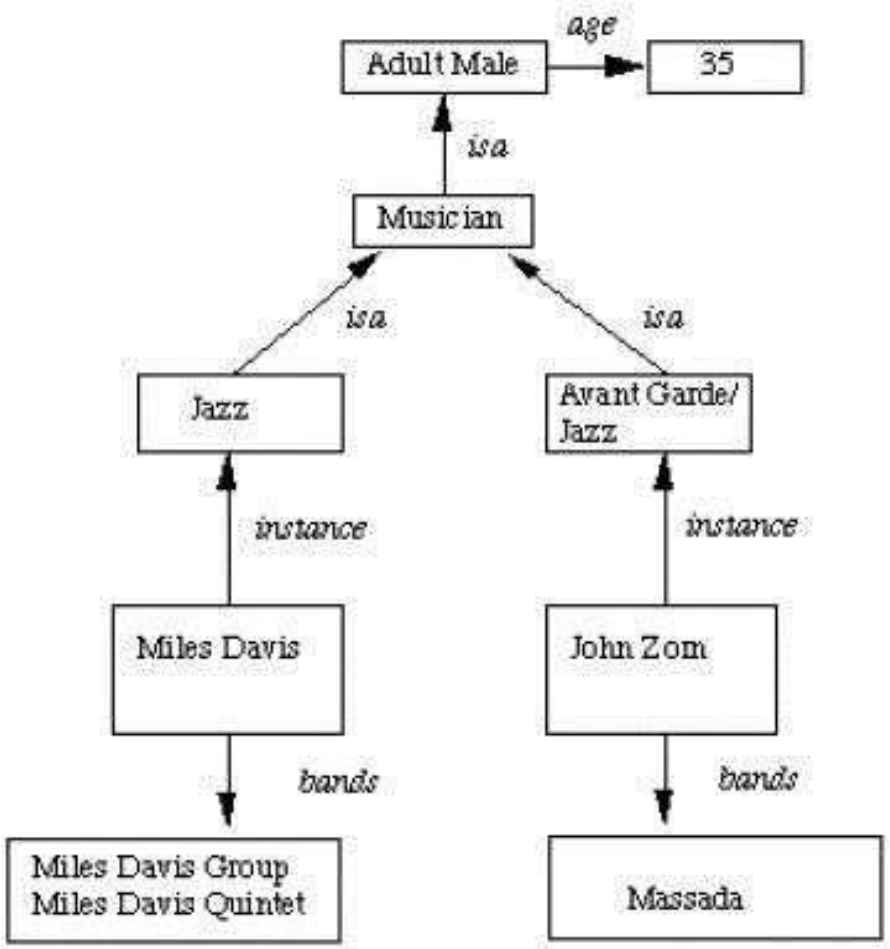
3. Inferential Efficiency

The ability to incorporate additional information into the knowledge structure that can be used to focus the attention of the inference mechanisms in the most promising direction.

Acquisitional Efficiency
Moreover, The ability to acquire new knowledge using automatic methods wherever possible rather than reliance on human intervention.

Player	Height	Weight	Bats - Throws
Aaron	6-0	180	Right - Right
Mays	5-10	170	Right - Right
Ruth	6-2	215	Left - Left
Williams	6-3	205	Left - Right

- : “Who is the heaviest player?”
- if a procedure for finding the heaviest player is provided, then these facts will enable that procedure to compute an answer.
- Moreover, We can ask things like who “bats – left” and “throws – right”. Inheritable Knowledge



- The inheritance is a powerful form of inference, but not adequate.
- Moreover, The basic KR (Knowledge Representation) needs to augment with inference mechanism.
- Property inheritance: The objects or elements of specific classes inherit attributes and values from more general classes.
- So, The classes organized in a generalized hierarchy.

- Arrows — the point from object to its value.
- This structure is known as a slot and filler structure, semantic network or a collection of frames.

The steps to retrieve a value for an attribute of an instance object:

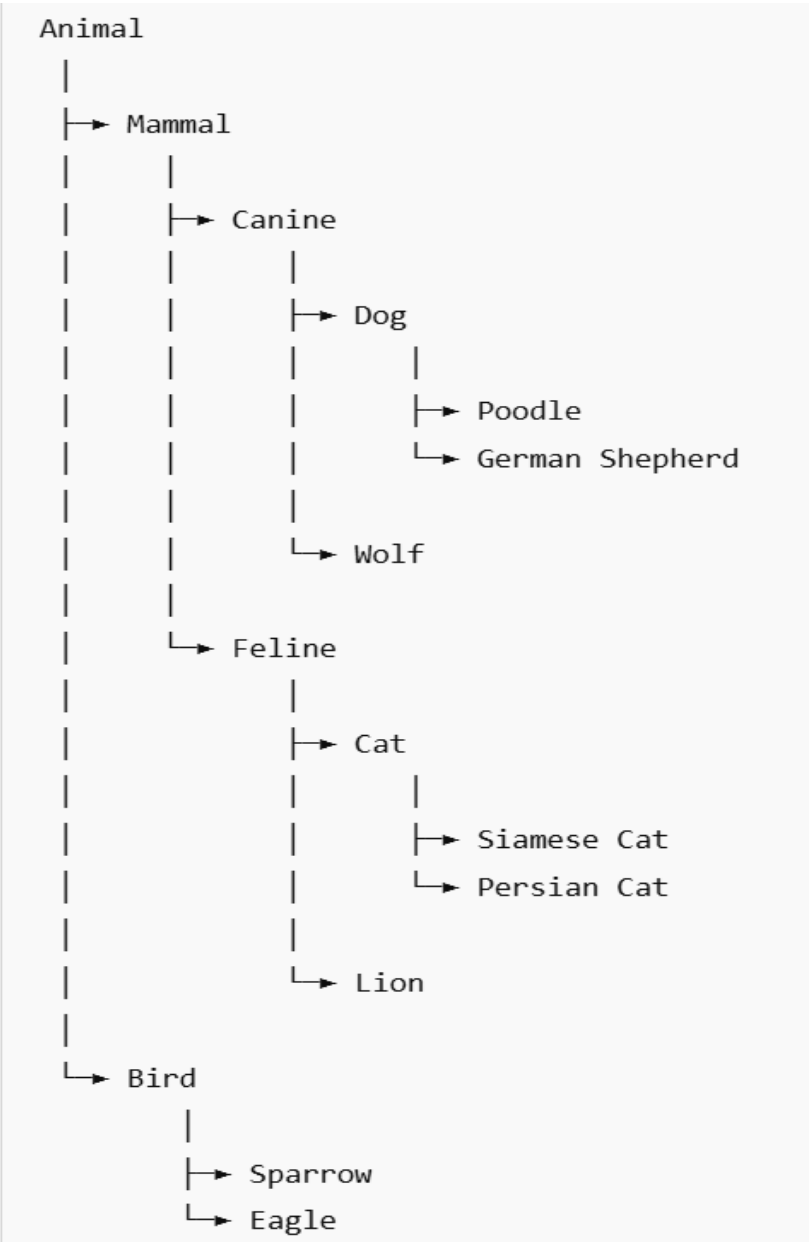
1. Find the object in the knowledge base
2. If there is a value for the attribute report it
3. Otherwise look for a value of an instance, if none fail
4. Also, Go to that node and find a value for the attribute and then report it
5. Otherwise, search through using is until a value is found for the attribute.

Inferential Knowledge

- This knowledge generates new information from the given information.
- This new information does not require further data gathering form source but does require analysis of the given information to generate new knowledge.

Procedural Knowledge

A representation in which the control information, to use the knowledge, embedded in the knowledge itself. For example, computer programs, directions, and recipes; these indicate specific use or implementation;



In this structure:

- **Poodle isa Dog:** The Poodle inherits characteristics from the broader category "Dog".
- **Dog isa Canine:** All dogs share common attributes defined by "Canine".
- **Canine isa Mammal:** The characteristics of mammals (like warm-bloodedness) are inherited by canines.
- **Mammal isa Animal:** All mammals are a subset of the broader category "Animal".

Similarly, the "Cat" node inherits from "Feline", which inherits from "Mammal", and so on. Each "isa" link creates a pathway where specific concepts inherit properties from more general ones, allowing the network to efficiently organize knowledge.

Logic

- Representation of knowledge and the reasoning -central to the entire field of artificial intelligence.
- The primary component of a knowledge-based agent is its knowledge-base.
- A knowledge-base is a set of sentences -expressed in a language called the knowledge representation language.
- Sentences represent some assertions about the world.
- There must mechanisms to derive new sentences from old ones.
- This process is known as inferencing or reasoning.
- Inference must obey the primary requirement that the new sentences should follow logically from the previous ones.
- *Logic* is the primary vehicle for representing and reasoning about knowledge.

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.

- A logic consists of two parts, a language and a method of reasoning.
- The logical language, in turn, has two aspects, syntax and semantics.
- Thus, to specify or define a particular logic, one needs to specify three things:
- **Syntax:**
  - The atomic symbols of the logical language, and the rules for constructing well-formed, non-atomic expressions (symbol structures) of the logic.
  - Syntax specifies the symbols in the language and how they can be combined to form sentences. Hence facts about the world are represented as sentences in logic.
- **Semantics:**
  - The meanings of the atomic symbols of the logic, and the rules for determining the meanings of non-atomic expressions of the logic.
  - It specifies what facts in the world a sentence refers to.
  - how you assign a truth value to a sentence based on its meaning in the world.
  - A **fact** is a claim about the world, and may be true or false.
- **Syntactic Inference Method:**
  - The rules for determining a subset of logical expressions, called theorems of the logic. It refers to mechanical method for computing (deriving) new (true) sentences from existing sentences.

Propositional Logic (PL)

A proposition is a statement, which in English would be a declarative sentence. Every proposition is either TRUE or FALSE.

Examples: **(a) The sky is blue., (b) Snow is cold. , (c) 12 * 12=144**

- Propositions are "sentences" , either true or false but not both.
- A sentence is smallest unit in propositional logic.
- If proposition is true, then truth value is "true" .
  If proposition is false, then truth value is "false" .

Example :

Sentence	Truth value	Proposition (Y/N)
"Grass is green"	"true"	Yes
"2 + 5 = 5"	"false"	Yes
"Close the door"	-	No
"Is it hot out side ?"	-	No
"x > 2" where x is variable	-	No (since x is not defined)
"x = x"	-	No (don't know what is "x" and "="; "3 = 3" or "air is equal to air" or "Water is equal to water" has no meaning)

■ Statement, Variables and Symbols

These and few more related terms, such as, *connective, truth value, contingencies, tautologies, contradictions, antecedent, consequent, argument* are explained below.

◇ Statement

Simple statements (sentences), TRUE or FALSE, that does not contain any other statement as a part, are basic propositions; lower-case letters, **p, q, r,** are symbols for simple statements.

Large, compound or complex statement are constructed from basic propositions by combining them with connectives.

◇ Connective or Operator

The connectives join simple statements into compounds, and joins compounds into larger compounds.

Table below indicates, the *basic connectives* and their symbols :

- listed in decreasing order of operation priority;
- operations with higher priority is solved first.

Example of a formula : **((((a Λ ¬b) V c → d) ↔ ¬ (a V c))**

Connectives and Symbols in decreasing order of operation priority

Connective	*Symbols*					Read as
assertion	P					"p is true"
negation	¬p	~	!		NOT	"p is false"
conjunction	p ∧ q	·	&&	&	AND	"both p and q are true"
disjunction	P v q	||	|		OR	"either p is true, or q is true, or both "
implication	p → q	⊃	⇒		if ..then	"if p is true, then q is true" " p implies q "
equivalence	↔	≡	⇔		if and only if	"p and q are either both true or both false"

Note : The propositions and connectives are the basic elements of propositional logic.

◇ Tautologies

A proposition that is always true is called a "tautology".

e.g., **(P v ¬P)** is always true regardless of the truth value of the proposition **P**.

◇ Contradictions

A proposition that is always false is called a "contradiction".

e.g., **(P ∧ ¬P)** is always false regardless of the truth value of the proposition **P**.

◇ Contingencies

A proposition is called a "contingency", if that proposition is neither a *tautology* nor a *contradiction* .

e.g., **(P v Q)** is a contingency.

◇ Antecedent, Consequent

These two are parts of conditional statements.

In the conditional statements, **p → q** , the

1st statement or **"if - clause"** (here **p**) is called **antecedent** ,

2nd statement or **"then - clause"** (here **q**) is called **consequent.**

- An argument is **valid**

  "if and only if" its corresponding conditional is a *tautology*.

- Two statements are **consistent**

  "if and only if" their conjunction is not a contradiction.

- Two statements are **logically equivalent**

  "if and only if" their truth table columns are identical;

  "if and only if" the statement of their equivalence using **" ≡ "** is a tautology.

◇ Truth Value

The truth value of a statement is its *TRUTH* or *FALSITY* ,

Example :

p is either *TRUE* or *FALSE*,

~p is either *TRUE* or *FALSE*,

p v q is either *TRUE* or *FALSE*, and so on.

use " **T** " or " **1** " to mean *TRUE*.

use " **F** " or " **0** " to mean *FALSE*

Truth table **defining the basic *connectives* :**

p	q	¬p	¬q	p ∧ q	p v q	p→q	p ↔ q	q→p
T	T	F	F	T	T	T	T	T
T	F	F	T	F	T	F	F	T
F	T	T	F	F	T	T	F	F
F	F	T	T	F	F	T	T	T

In propositional logic (PL) an user defines a set of propositional symbols, like *P* and *Q*. User defines the semantics of each of these symbols. For example,

- P means "It is hot"
- Q means "It is humid"
- R means "It is raining"
-

- A **sentence** (also called a formula or well-formed formula or wff) is defined as:
  1. A symbol
  2. If S is a sentence, then ~S is a sentence, where "~" is the "not" logical operator
  3. If S and T are sentences, then (S v T), (S ^ T), (S => T), and (S <=> T) are sentences, where the four logical connectives correspond to "or," "and," "implies," and "if and only if," respectively
  4. A finite number of applications of (1)-(3)

- Examples of PL sentences:
  - (P ^ Q) => R (here meaning "If it is hot and humid, then it is raining")
  - Q => P (here meaning "If it is humid, then it is hot")
  - Q (here meaning "It is humid.")

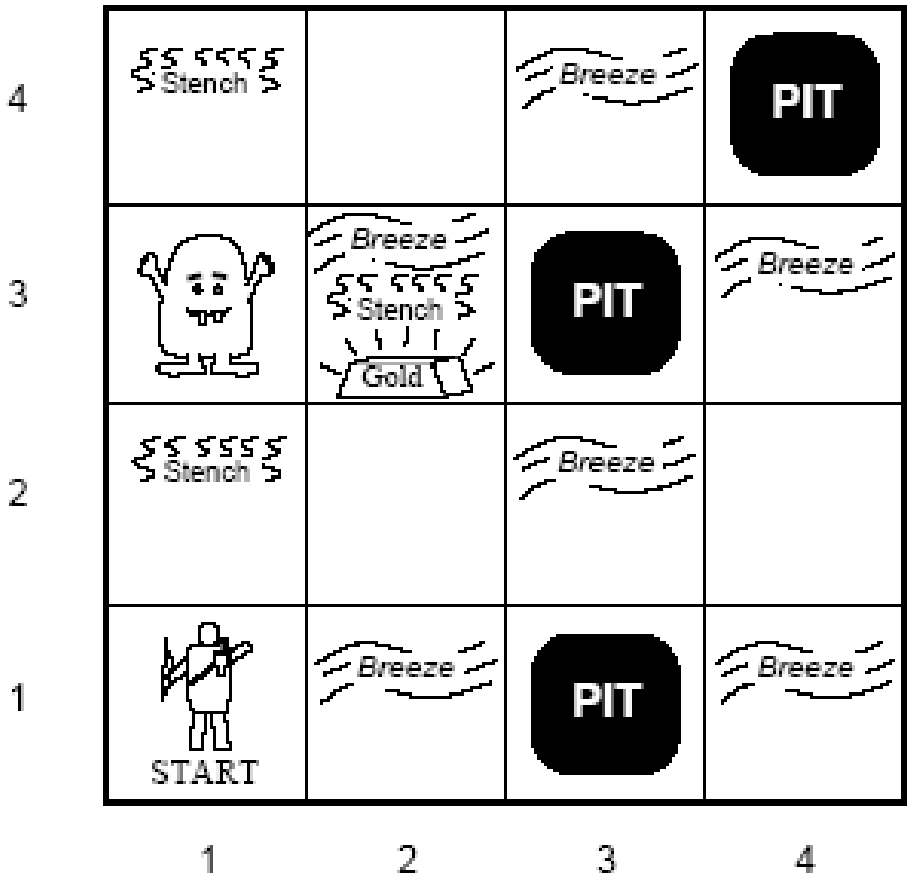
Propositional logic: Syntax

- Propositional logic is the simplest logic – illustrates basic ideas

- The proposition symbols P_1, P_2 etc are sentences
  - If S is a sentence, $\neg$S is a sentence (negation)
  - If S_1 and S_2 are sentences, $S_1 \wedge S_2$ is a sentence (conjunction)
  - If S_1 and S_2 are sentences, $S_1 \vee S_2$ is a sentence (disjunction)
  - If S_1 and S_2 are sentences, $S_1 \Rightarrow S_2$ is a sentence (implication)
  - If S_1 and S_2 are sentences, $S_1 \Leftrightarrow S_2$ is a sentence (biconditional)

Wumpus World

- Performance Measure
  - Gold +1000, Death – 1000
  - Step -1, Use arrow -10
- Environment
  - Square adjacent to the Wumpus are smelly
  - Squares adjacent to the pit are breezy
  - Glitter iff gold is in the same square
  - Shooting kills Wumpus if you are facing it
  - Shooting uses up the only arrow
  - Grabbing picks up the gold if in the same square
- Actuators
  - Left turn, right turn, forward, grab, shoot
- Sensors
  - Breeze, glitter, and smell



Wumpus World Sentences

- Let $P_{i,j}$ be True if there is a pit in [i,j]
- Let $B_{i,j}$ be True if there is a breeze in [i,j]

- $\neg P_{1,1}$
- $\neg B_{1,1}$
- $B_{2,1}$

- "Pits cause breezes in adjacent squares"

$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

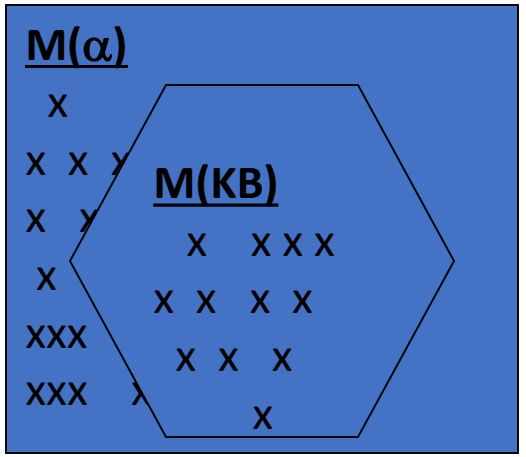
$B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,1} \vee P_{3,1})$

- A square is breezy if and only if there is an adjacent pit

Model

- A model is a formally structured world with respect to which truth can be evaluated
  - M is a model of sentence α if α is true in m

- Then KB $\models \alpha$ if $M(KB) \subseteq M(\alpha)$



Example:

x + y = 4 entails 4 = x + y

Entailment

- Sentence P **entails** sentence Q, written P |= Q, means that whenever P is True, so is Q. In other words, all models of P are also models of Q

Example: Entailment

Show that: $p \wedge (p \Rightarrow q) \models q$

Proof: For any model M in which $p \wedge (p \Rightarrow q)$ holds then we know that p holds in M and $p \Rightarrow q$ holds in M. Since p holds in M then since $p \Rightarrow q$ holds in M, q must hold in M. Therefore q holds in every model that $p \wedge (p \Rightarrow q)$ holds and so $p \wedge (p \Rightarrow q) \models q$.

This section covers standard patterns of inference that can be applied to derive chains of conclusions that lead to the desired goal. These patterns of inference are called **inference rules.** The best-known rule is called **Modus Ponens** and is written as follows:

$$\frac{\alpha \Rightarrow \beta, \quad \alpha}{\beta}$$

The notation means that, whenever any sentences of the form a $\Rightarrow \beta$ and a are given, then the sentence β can be inferred. For example, if $(WumpusAhead \wedge WumpusAlive) \Rightarrow Shoot$ and $(WumpusAhead \wedge WumpusAlive)$ are given, then *Shoot* can be inferred.

Another useful inference rule is **And-Elimination,** which says that, from a conjunction, any of the conjuncts can be inferred:

$$\frac{\alpha \wedge \beta}{\alpha}$$

For example, from $(WumpusAhead \wedge WumpusAlive)$, *WumpusAlive* can be inferred.

By considering the possible truth values of a and β, one can show easily that Modus Ponens and And-Elimination are sound once and for all. These rules can then be used in any particular instances where they apply, generating sound inferences without the need for enumerating models.

Predicate Logic

The propositional logic, is not powerful enough for all types of assertions;

Example : The assertion **"x > 1"**, where **x** is a variable, is not a proposition because it is neither true nor false unless value of **x** is defined.

For **x > 1** to be a proposition ,

- either we substitute a specific number for **x** ;
- or change it to something like

  "There is a number x for which x > 1 holds";
- or **"For every number x, x > 1 holds"**.

Consider example :

" All men are mortal.

Socrates is a man.

Then Socrates is mortal" ,

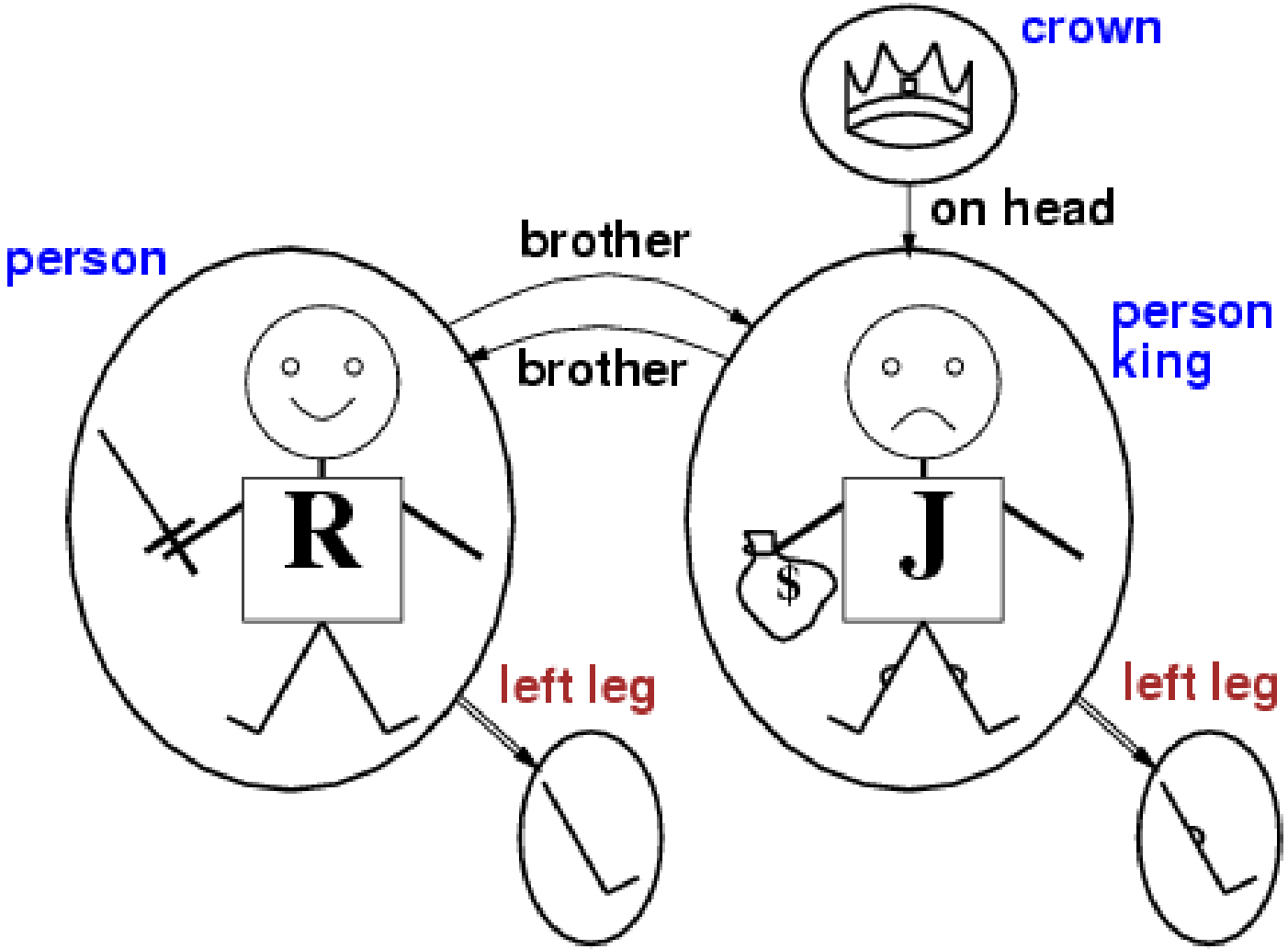
These cannot be expressed in propositional logic as a finite and logically valid argument (formula).

We need languages : that allow us to describe properties (*predicates*) of objects, or a relationship among objects represented by the variables .

Predicate logic satisfies the requirements of a language.

- *Predicate logic* is powerful enough for expression and reasoning.
- *Predicate logic* is built upon the ideas of *propositional logic.*

Models for FOL: Example



Syntax of FOL

- **Constants:** John, Sally, 2, ...
- **Variables:** x, y, a, b,...
- **Predicates:** Person(John), Siblings(John, Sally), IsOdd(2), ...
- **Functions:** MotherOf(John), Sqrt(x), ...
- **Connectives:** $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$
- **Equality:** =
- **Quantifiers:** $\forall, \exists$

- **Term:** Constant or Variable or Function($Term_1$, ... , $Term_n$)
- **Atomic sentence:** Predicate($Term_1$, ... , $Term_n$) or $Term_1 = Term_2$
- **Complex sentence:** made from atomic sentences using connectives and quantifiers

Semantics of FOL

- Sentences are true with respect to a model and an interpretation
- Model contains objects (domain elements) and relations among them
- Interpretation specifies referents for

  constant symbols → objects

  predicate symbols → relations

  function symbols → functional relations

- An atomic sentence **Predicate(Term$_1$, ... , Term$_n$)** is true iff the objects referred to by **Term$_1$, ... , Term$_n$** are in the relation referred to by predicate

Sentences in First-Order Logic

- An atomic sentence is simply a predicate applied to a set of terms.

  Owns(John,Car1)
  Sold(John,Car1,Fred)

  Semantics is True or False depending on the interpretation, i.e. is the predicate true of these arguments.

- The standard propositional connectives ($\vee$ $\neg$ $\wedge$ $\Rightarrow$ $\Leftrightarrow$) can be used to construct complex sentences:

  Owns(John,Car1) $\vee$ Owns(Fred, Car1)
  Sold(John,Car1,Fred) $\Rightarrow$ $\neg$Owns(John, Car1)

  Semantics same as in propositional logic.

Quantifiers

- Once we have a logic that allows objects, it is only natural to want to express properties of entire collections of objects, instead of enumerating the objects by name. Quantifiers let us do this.
- First-order logic contains two standard quantifiers, called *Universal* and *existential*

Universal quantification naturally uses implication:

- "All kings are persons," is written in first-order logic as
- $\forall x\ King(x) \Rightarrow Person(x)$
- ∀is usually pronounced "For all . . .".
- Thus, the sentence says, "For all *x,* if *x* is a king, then *x*is a person." The symbol x is called a **variable**. By convention, variables are lowercase letters.
- A variable is a term all by itself, and as such can also serve as the argument of a function-for example, *LeftLeg(x).*
- A term with no variables is called a **ground term.**
- Intuitively, the sentence ∀*x P,* where *P* is any logical expression, says that *P* is true for every object *x.*
- More precisely, ∀*x* P is true in a given. model under a given interpretation if P is true in all possible **extended interpretations** constructed from the given interpretation, where each extended interpretation specifies a domain element to which x refers.

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

  ∃ *x Crown(x) A OnHead (x, John)* .

- ∃ *x*is pronounced "There exists an x such that . . ."or "For some x . . .".
- Intuitively, the sentence ∃x P says that *P* is true for at least one object *x.*
- More precisely, ∃x P is true in a given model under a given interpretation if P is true in *at* least *one* extended interpretation that assigns *x* to a domain element.
- For our example, this means that at least one of the following must be true:
  - Richard is a crown ∧ Richard is on John's head;
  - King John is a crown ∧ King John is on John's head;
  - Richard's left leg is a crown ∧ Richard's left leg is on John's head;
  - John's left leg is a crown ∧ John's left leg is on John's head;
  - The crown is a crown ∧ the crown is on John's head.

- The fifth assertion is true in the model, so the original existentially quantified sentence is true in the model.
- Notice that, by our definition, the sentence would also be true in a model in which King John was wearing two crowns.
- This is entirely consistent with the original sentence "King John has a crown on his head."
- Just as $\Rightarrow$ appears to be the natural connective to use with $\forall$, $\wedge$ is the natural connective to use with $\exists$. Using $\wedge$ as the main connective with $\forall$ led to an overly strong statement in the example in the previous section; using $\Rightarrow$ with $\exists$ usually leads to a very weak statement, indeed.

Nested quantifiers

We will often want to express more complex sentences using multiple quantifiers. The simplest case is where the quantifiers are of the same type. For example, "Brothers are siblings" can be written as

$$\forall x \, \forall y \, Brother(x, y) \Rightarrow Sibling(x, y) \; .$$

Consecutive quantifiers of the same type can be written as one quantifier with several variables.

For example, to say that siblinghood is a symmetric relationship, we can write

$$\forall x, y \, Sibling(x, y) \Leftrightarrow Sibing(y, z) \; .$$

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

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

The order of quantification is therefore very important. It becomes clearer if we insert parentheses.

$\forall x \, (\exists y \, Loves \, (x, y))$ says that *everyone* has a particular property, namely, the property that they love someone. 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.

Equality

- $term_1 = term_2$ is true under a given interpretation if and only if $term_1$ and $term_2$ refer to the same object

- E.g., definition of *Sibling* in terms of *Parent*:

  $\forall x,y\ Sibling(x,y) \Leftrightarrow [\neg(x = y) \wedge \exists m,f \neg (m = f) \wedge$
  $Parent(m,x) \wedge Parent(f,x) \wedge Parent(m,y) \wedge Parent(f,y)]$

Equality

- Can include equality as a primitive predicate in the logic, or require it to be introduced and axiomitized as the **identity relation**.

- Useful in representing certain types of knowledge:

  $\exists x \exists y(Owns(Mary, x) \wedge Cat(x) \wedge Owns(Mary,y) \wedge Cat(y)$
  $\wedge \neg(x=y))$

  Mary owns two cats. Inequality needed to insure x and y are distinct.

  $\forall x\ \exists y\ married(x, y) \wedge \forall z(married(x,z) \Rightarrow y=z)$

  Everyone is married to exactly one person. Second conjunct is needed to guarantee there is only one unique spouse.

- Allows statements about entire collections of objects rather than having to enumerate the objects by name.

- Universal quantifier: ∀x
  Asserts that a sentence is true for all values of variable x

  ∀x Loves(x, FOPC)
  ∀x Whale(x) ⇒ Mammal(x)
  ∀x Grackles(x) ⇒ Black(x)
  ∀x (∀y Dog(y) ⇒ Loves(x,y)) ⇒ (∀z Cat(z) ⇒ Hates(x,z))

- **∀x P(x)**

- Example: "Everyone at SASTRA is smart"
  ∀x At(x,SASTRA) ⇒ Smart(x)

- Roughly speaking, equivalent to the conjunction of all possible instantiations of the variable:
  [At(Joel, SASTRA) ⇒ Smart(Joel)] ∧ ...
  [At(Hari, SASTRA) ⇒ Smart(Hari)] ∧ ...

- **∀x P(x)** is true in a model m iff **P(x)** is true with **x** being each possible object in the model

- Existential quantifier: ∃
  Asserts that a sentence is true for at least one value of a variable x

  ∃x Loves(x, FOPC)
  ∃x(Cat(x) ∧ Color(x,Black) ∧ Owns(Mary,x))
  ∃x(∀y Dog(y) ⇒ Loves(x,y)) ∧ (∀z Cat(z) ⇒ Hates(x,z))

Properties of quantifiers

- $\forall x\ \forall y$ is the same as $\forall y\ \forall x$
- $\exists x\ \exists y$ is the same as $\exists y\ \exists x$
- $\exists x\ \forall y$ is not the same as $\forall y\ \exists x$
  - $\exists x\ \forall y$ Loves(x,y)
  - “There is a person who loves everyone in the world”
  - $\forall y\ \exists x$ Loves(x,y)
  - “Everyone in the world is loved by at least one person”
- Quantifier duality: each can be expressed using the other
- $\forall x$ Likes(x,IceCream) $\quad\quad \neg\exists x\ \neg$Likes(x,IceCream)
- $\exists x$ Likes(x,Broccoli) $\quad\quad \neg\forall x\ \neg$Likes(x,Broccoli)

Relation Between Quantifiers

- Universal and existential quantification are logically related to each other:

$\forall x \neg \text{Love}(x,\text{Saddam}) \Leftrightarrow \neg \exists x \text{ Loves}(x,\text{Saddam})$

$\forall x \text{ Love}(x,\text{Princess-Di}) \Leftrightarrow \neg \exists x \neg \text{Loves}(x,\text{Princess-Di})$

- General Identities
  - $\forall x \neg P \Leftrightarrow \neg \exists x P$
  - $\neg \forall x P \Leftrightarrow \exists x \neg P$
  - $\forall x P \Leftrightarrow \neg \exists x \neg P$
  - $\exists x P \Leftrightarrow \neg \forall x \neg P$

  - $\forall x P(x) \wedge Q(x) \Leftrightarrow \forall x P(x) \wedge \forall x Q(x)$
  - $\exists x P(x) \vee Q(x) \Leftrightarrow \exists x P(x) \vee \exists x Q(x)$

Because ∀is really a conjunction over the universe of objects and ∃is a disjunction that they obey De Morgan's rules. The De Morgan rules for quantified and unquantified sentences are as follows:

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

Thus, Quantifiers are important in terms of readability.

$$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha) \quad \text{commutativity of } \wedge$$
$$(\alpha \vee \beta) \equiv (\beta \vee \alpha) \quad \text{commutativity of } \vee$$
$$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma)) \quad \text{associativity of } \wedge$$
$$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma)) \quad \text{associativity of } \vee$$
$$\neg(\neg\alpha) \equiv \alpha \quad \text{double-negation elimination}$$
$$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha) \quad \text{contraposition}$$
$$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta) \quad \text{implication elimination}$$
$$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) \quad \text{biconditional elimination}$$
$$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta) \quad \text{de Morgan}$$
$$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta) \quad \text{de Morgan}$$
$$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) \quad \text{distributivity of } \wedge \text{ over } \vee$$
$$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) \quad \text{distributivity of } \vee \text{ over } \wedge$$

Well formed Formulas:

- Consider the following example that shows the use of predicate logic as a way of representing knowledge.
- Marcus was a man.
- Marcus was a Pompeian.
- All Pompeians were Romans.
- Caesar was a ruler.
- Also, All Pompeians were either loyal to Caesar or hated him.
- Everyone is loyal to someone.
- People only try to assassinate rulers they are not loyal to.
- Marcus tried to assassinate Caesar.

The facts described by these sentences can be represented as a set of well-formed formulas (wffs) as follows:

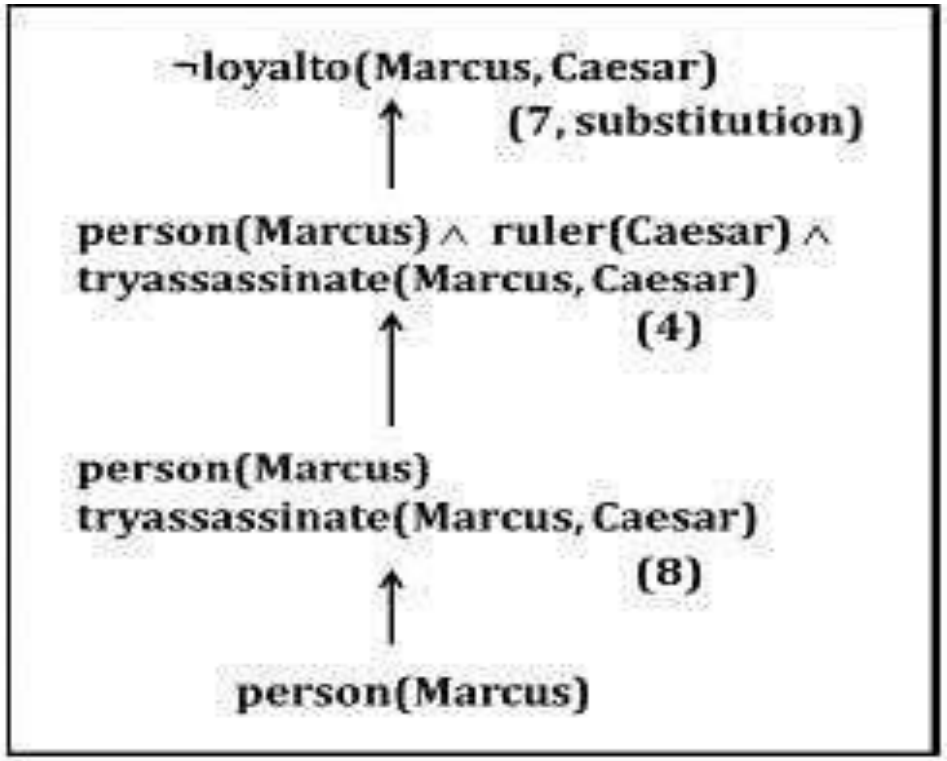
1. Marcus was a man.
   man(Marcus)
2. Marcus was a Pompeian.
   Pompeian(Marcus)
3. All Pompeians were Romans.
   $\forall x$: Pompeian(x) $\rightarrow$ Roman(x)
4. Caesar was a ruler.
   ruler(Caesar)

5. All Pompeians were either loyal to Caesar or hated him.
    - inclusive-or
        - $\forall x$: Roman(x) $\rightarrow$ loyalto(x, Caesar) $\vee$ hate(x, Caesar)
    - exclusive-or
        - $\forall x$: Roman(x) $\rightarrow$ (loyalto(x, Caesar) $\wedge\neg$ hate(x, Caesar)) $\vee$
        - ($\neg$loyalto(x, Caesar) $\wedge$ hate(x, Caesar))
6. Everyone is loyal to someone.
    - $\forall x$: $\exists y$: loyalto(x, y)
7.People only try to assassinate rulers they are not loyal to.

    $\forall x$: $\forall y$: person(x) $\wedge$ ruler(y) $\wedge$ tryassassinate(x, y) $\rightarrow$ $\neg$loyalto(x, y)
8. Marcus tried to assassinate Caesar.
    - tryassassinate(Marcus, Caesar)

Now suppose if we want to use these statements to answer the question: Was Marcus loyal to Caesar?

- Also, Now let’s try to produce a formal proof, reasoning backward from the desired goal: ¬ loyalto(Marcus, Caesar)
- In order to prove the goal, we need to use the rules of inference to transform it into another goal (or possibly a set of goals) that can, in turn, transformed, and so on, until there are no unsatisfied goals remaining.



Representing Instance and ISA Relationships

- Specific attributes instance and isa play an important role particularly in a useful form of reasoning called property inheritance.
- The predicates instance and isa explicitly captured the relationships they used to express, namely class membership and class inclusion.

(first 5), class membership represented with unary predicates (such as Roman), each of which corresponds to a class.

- Asserting that P(x) is true is equivalent to asserting that x is an instance (or element) of P.

1. Man(Marcus).
2. Pompeian(Marcus).
3. $\forall x$: Pompeian(x) $\rightarrow$ Roman(x).
4. ruler(Caesar).
5. $\forall x$: Roman(x) $\rightarrow$ loyalto(x, Caesar) $\vee$ hate(x, Caesar).

1. instance(Marcus, man).
2. instance(Marcus, Pompeian).
3. $\forall x$: instance(x, Pompeian) $\rightarrow$ instance(x, Roman).
4. instance(Caesar, ruler).
5. $\forall x$: instance(x, Roman). $\rightarrow$ loyalto(x, Caesar) $\vee$ hate(x, Caesar).

- The second part of the figure contains representations that use the instance predicate explicitly.
- The predicate instance is a binary one, whose first argument is an object and whose second argument is a class to which the object belongs.
- But these representations do not use an explicit isa predicate.
- Instead, subclass relationships, such as that between Pompeians and Romans, described as shown in sentence 3.
- The implication rule states that if an object is an instance of the subclass Pompeian then it is an instance of the superclass Roman.
- Note that this rule is equivalent to the standard set-theoretic definition of the subclass- superclass relationship.

1. Man(Marcus).
2. Pompeian(Marcus).
3. $\forall x$: Pompeian(x) $\rightarrow$ Roman(x).
4. ruler(Caesar).
5. $\forall x$: Roman(x) $\rightarrow$ loyalto(x, Caesar) $\vee$ hate(x, Caesar).

- • The third part contains representations that use both the instance and isa predicates explicitly.
- • The use of the isa predicate simplifies the representation of sentence 3, but it requires that one additional axiom (shown here as number 6) be provided.

1. instance(Marcus, man).
2. instance(Marcus, Pompeian).
3. isa(Pompeian, Roman)
4. instance(Caesar, ruler).
5. $\forall x$: instance(x, Roman). $\rightarrow$ loyalto(x, Caesar) $\vee$ hate(x, Caesar).
6. $\forall x$: $\forall y$: $\forall z$: instance(x, y) $\wedge$ isa(y, z) $\rightarrow$ instance(x, z).

Computable Functions and Predicates

- To express simple facts, such as the following greater-than and less-than relationships: gt(1,O) It(0,1) gt(2,1) It(1,2) gt(3,2) It(2,3)
- It is often also useful to have computable functions as well as computable predicates. Thus we might want to be able to evaluate the truth of gt(2 + 3,1)
- To do so requires that we first compute the value of the plus function given the arguments 2 and 3, and then send the arguments 5 and 1 to gt.

Consider the following set of facts, again involving Marcus:

1) Marcus was a man.

man(Marcus)

2) Marcus was a Pompeian.

Pompeian(Marcus)

3) Marcus was born in 40 A.D.

born(Marcus, 40)

4) All men are mortal.

x: man(x) → mortal(x)

5) All Pompeians died when the volcano erupted in 79 A.D.

erupted(volcano, 79) ∧ ∀ x : [Pompeian(x) → died(x, 79)]

6) No mortal lives longer than 150 years.

x: t1: At2: mortal(x) born(x, t1) gt(t2 – t1,150) → died(x, t2)

7) It is now 1991.

now = 1991

So, Above example shows how these ideas of computable functions and predicates can be useful. It also makes use of the notion of equality and allows equal objects to be substituted for each other whenever it appears helpful to do so during a proof.

So, Now suppose we want to answer the question “Is Marcus alive?”

- The statements suggested here, there may be two ways of deducing an answer.
- Either we can show that Marcus is dead because he was killed by the volcano or we can show that he must be dead because he would otherwise be more than 150 years old, which we know is not possible.

For example, our statements talk about dying, but they say nothing that relates to being alive, which is what the question is asking.

So we add the following facts: 8) Alive means not dead.

- 9) If someone dies, then he is dead at all later times.
- x: t1: At2: died(x, t1) gt(t2, t1) → dead(x, t2)
  So, Now let's attempt to answer the question "Is Marcus alive?" by proving: ¬ alive(Marcus, now)

- **Propositional Vs First Order Inference**
- Earlier inference in first order logic is performed with Propositionalization which is a process of converting the Knowledgebase present in First Order logic into Propositional logic and on that using any inference mechanisms of propositional logic are used to check inference.
  Inference rules for quantifiers:
- There are some Inference rules that can be applied to sentences with quantifiers to obtain sentences without quantifiers. These rules will lead us to make the conversion.

Inference rules for quantifiers

Let us begin with universal quantifiers. Suppose our knowledge base contains the standard folkloric axiom stating that all greedy kings are evil:

$$\forall x \;\; King(x) \wedge Greedy(x) \Rightarrow Evil(x) .$$

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

$$King(John) \wedge Greedy(John) \Rightarrow Evil(John).$$
$$King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard) .$$
$$King(Father(John)) \wedge Greedy(Father(John)) \Rightarrow Evil(Father(John)).$$

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. [1] To write out the inference rule formally, we use the notion of **substitutions** introduced in Section 8.3. Let $\text{SUBST}(\theta, a)$ denote the result of applying the substitution θ to the sentence a. Then the rule is written

$$\frac{\forall v \;\; a}{\text{SUBST}(\{v/g\}, \alpha)}$$

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

Existential instantiation (EI)

- An existentially quantified sentence entails the instantiation of that sentence with a new constant:

$$\frac{\exists v\ P(v)}{SUBST(\{v/C\}, P(v))}$$

for any sentence P, variable v, and constant C that does not appear elsewhere in the knowledge base

- E.g., $\exists x\ Crown(x) \wedge OnHead(x,John)$ yields:

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

provided C_1 is a new constant symbol, called a *Skolem constant*

Mathematics provides a nice example: suppose we discover that there is a number that is a little bigger than 2.71828 and that satisfies the equation $d(x^y)/dy = x^y$ for x. We can give this number a name, such as e, but it would be a mistake to give it the name of an existing object, such as $\prod$. In logic, the new name is called a **Skolem constant.** Existential Instantiation is a special case of a more general process called **skolemization.**

Reduction to propositional inference

Once we have rules for inferring non quantified sentences from quantified sentences, it becomes possible to reduce first-order inference to propositional inference. For example, suppose our knowledge base contains just the sentences

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

Then we apply UI to the first sentence using all possible ground term substitutions from the vocabulary of the knowledge base-in this case, {xl John) and {x/Richard). We obtain

$$King(John) \ A \ Greedy(John) \ \Rightarrow \ \ Evil(John),$$
$$King(Richard) \ A \ Greedy(Richard) \ \Rightarrow \ \ Evil(Richard)$$

We discard the universally quantified sentence. Now, the knowledge base is essentially propositional if we view the ground atomic sentences-King (John), Greedy (John), and Brother (Richard, John) as proposition symbols. Therefore, we can apply any of the complete propositional algorithms to obtain conclusions such as Evil (John).

A simple forward-chaining algorithm:

- Starting from the known facts, it triggers all the rules whose premises are satisfied, adding their conclusions of the known facts
- The process repeats until the query is answered or no new facts are added. Notice that a fact is not "new" if it is just renaming of a known fact.

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

- Prove that Col. West is a criminal

Example knowledge base

We will use our crime problem to illustrate how FOL-FC-ASK works. The implication sentences are (1), (4), (5), and (6). Two iterations are required:

- On the first iteration, rule (1) has unsatisfied premises.
  Rule (4) is satisfied with {x/Ml), and Sells (West, M1, Nono) is added.
  Rule (5) is satisfied with {x/M1) and Weapon (M1) is added.
  Rule (6) is satisfied with {x/Nono}, and Hostile (Nono) is added.

- On the second iteration, rule (1) is satisfied with {x/West, Y/MI, z /Nono), and Criminal (West) is added.

1. It is a crime for an American to sell weapons to hostile nations:
   $American(x) \wedge Weapon(y) \wedge Sells(x,y,z) \wedge Hostile(z) \Rightarrow Criminal(x)$

Nono has some missiles

2. $\exists x\ Owns(Nono,x) \wedge Missile(x)$

3. $Missile(M_1)$

   $Owns(Nono,M_1) \wedge Missile(M_1)$

4. All of its missiles were sold to it by Colonel West
   $Missile(x) \wedge Owns(Nono,x) \Rightarrow Sells(West,x,Nono)$

5. Missiles are weapons:
   $Missile(x) \Rightarrow Weapon(x)$

6. An enemy of America counts as "hostile":
   $Enemy(x,America) \Rightarrow Hostile(x)$

7. West is American
   $American(West)$

8. The country Nono is an enemy of America
   $Enemy(Nono,America)$

Forward chaining proof

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

Owns(Nono,M_1) ∧ Missile(M_1)

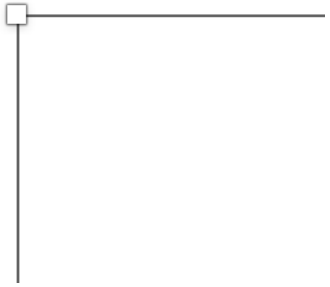
Missile(x) ∧ Owns(Nono,x) ⇒ Sells(West,x,Nono)

Missile(x) ⇒ Weapon(x)

Enemy(x,America) ⇒ Hostile(x)

American(West)

Enemy(Nono,America)



On the first iteration, rule (1) has unsatisfied premises.

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

Forward chaining proof

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

2 3 Owns(Nono,M_1) ∧ Missile(M_1)

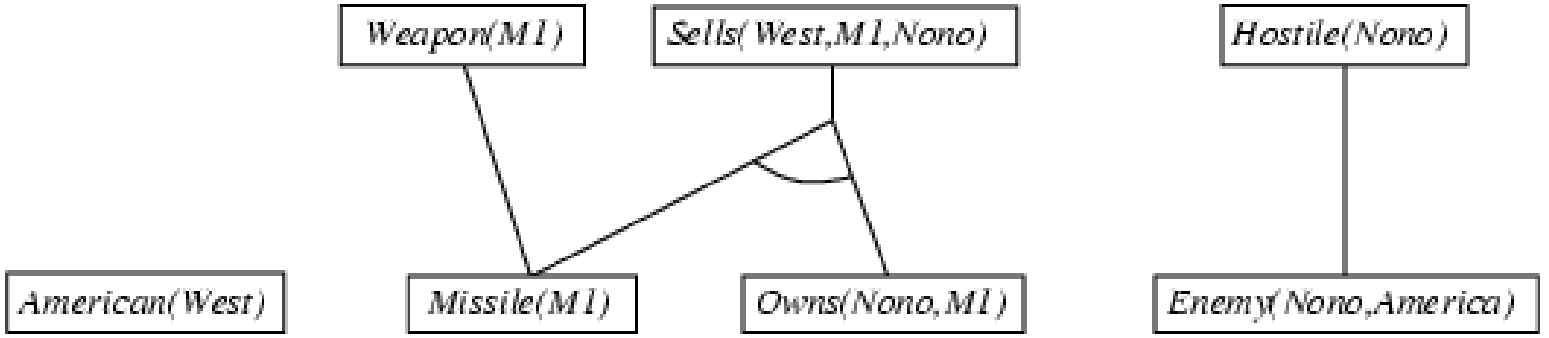
4 Missile(x) ∧ Owns(Nono,x) ⇒ Sells(West,x,Nono)

5 Missile(x) ⇒ Weapon(x)

6 Enemy(x,America) ⇒ Hostile(x)

7 American(West)

8 Enemy(Nono,America)



Rule (4) is satisfied with {x/Ml), and Sells (West, M1, Nono) is added.

Rule (5) is satisfied with {x/M1) and Weapon (M1) is added.

Forward chaining proof

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

2 3 Owns(Nono,M_1) ∧ Missile(M_1)

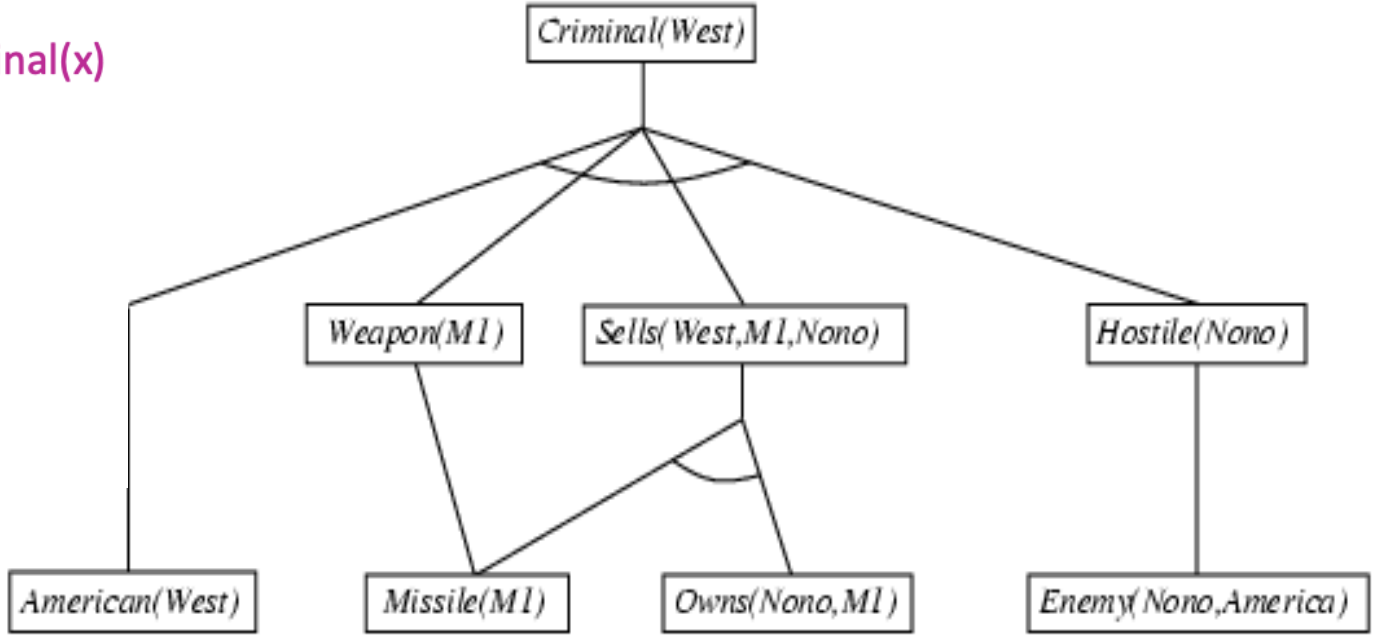
4 Missile(x) ∧ Owns(Nono,x) ⇒ Sells(West,x,Nono)

5 Missile(x) ⇒ Weapon(x)

6 Enemy(x,America) ⇒ Hostile(x)

7 American(West)

8 Enemy(Nono,America)



Rule (6) is satisfied with {x/Nono}, and Hostile (Nono) is added.

- On the second iteration, rule (1) is satisfied with {x/West, Y/MI, z /Nono), and Criminal (West) is added.

FC Algorithm

function FOL-FC-ASK(KB, α) **returns** a substitution or *false*

  repeat until *new* is empty
    $new \leftarrow \{\}$
    for each sentence r **in** KB **do**
      $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow$ STANDARDIZE-APART(r)
      for each θ such that $(p_1 \wedge \ldots \wedge p_n)\theta = (p'_1 \wedge \ldots \wedge p'_n)\theta$
          for some $p'_1, \ldots, p'_n$ in KB
        $q' \leftarrow$ SUBST(θ, q)
        if q' is not a renaming of a sentence already in KB or new **then do**
          add q' to new
          $\phi \leftarrow$ UNIFY(q', α)
          if ϕ is not *fail* **then return** ϕ
    add new to KB
  return *false*

Backward chaining example

Criminal(West)

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

2 3 Owns(Nono,M_1) ∧ Missile(M_1)

4 Missile(x) ∧ Owns(Nono,x) ⇒ Sells(West,x,Nono)

5 Missile(x) ⇒ Weapon(x)

6 Enemy(x,America) ⇒ Hostile(x)

7 American(West)

8 Enemy(Nono,America)

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

It is called with a list of goals containing the original query, and returns the set of all substitutions satisfying the query.

And from the goal fact, we will infer other facts, and at last, we will prove those facts true. So our goal fact is ”West is Criminal," so following is the predicate of it.

Backward chaining example

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

2 3 Owns(Nono,M_1) ∧ Missile(M_1)

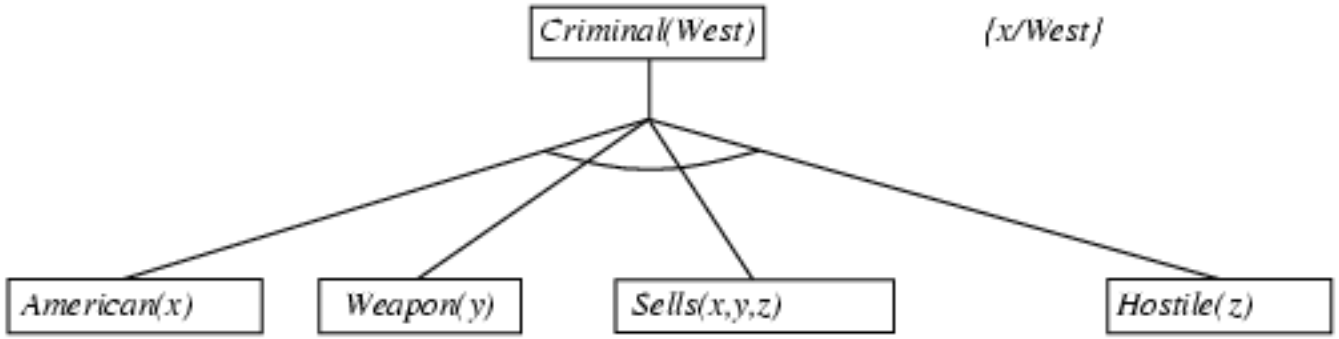
4 Missile(x) ∧ Owns(Nono,x) ⇒ Sells(West,x,Nono)

5 Missile(x) ⇒ Weapon(x)

6 Enemy(x,America) ⇒ Hostile(x)

7 American(West)

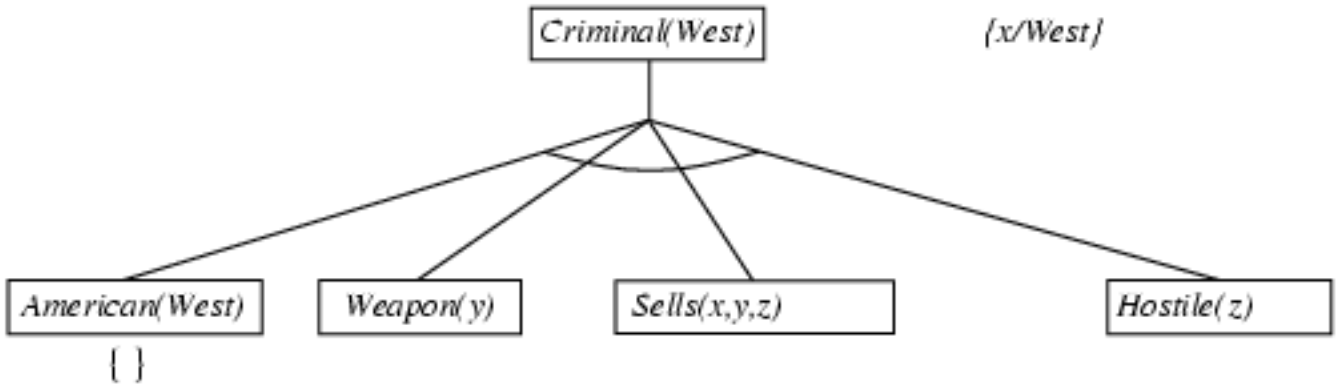
8 Enemy(Nono,America)



At the second step, we will infer other facts form goal fact which satisfies the rules. So as we can see in Rule-1, the goal predicate Criminal (West) is present with substitution {West /P}. So we will add all the conjunctive facts below the first level and will replace x with West.
Here we can see American (West) is a fact, so it is proved here.

Backward chaining example

1. $American(x) \wedge Weapon(y) \wedge Sells(x,y,z) \wedge Hostile(z) \Rightarrow Criminal(x)$
2. $Owns(Nono,M_1) \wedge Missile(M_1)$
3. $Missile(x) \wedge Owns(Nono,x) \Rightarrow Sells(West,x,Nono)$
4. $Missile(x) \Rightarrow Weapon(x)$
5. $Enemy(x,America) \Rightarrow Hostile(x)$
6. $American(West)$
7. $Enemy(Nono,America)$



Step-3:t At step-3, we will extract further fact Missile(x) which infer from Weapon(x), as it satisfies Rule-(5). Weapon (x) is also true with the substitution of a constant M1 at y.

Backward chaining example

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

2 3 Owns(Nono,M_1) ∧ Missile(M_1)

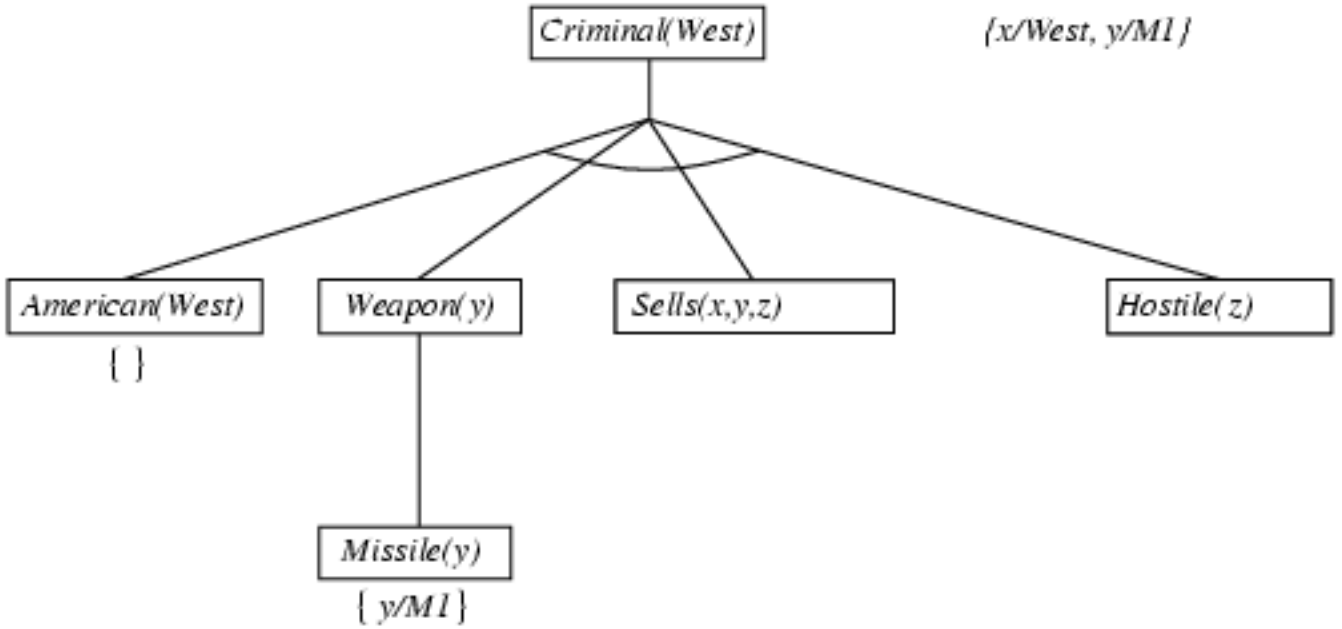
4 Missile(x) ∧ Owns(Nono,x) ⇒ Sells(West,x,Nono)

5 Missile(x) ⇒ Weapon(x)

6 Enemy(x,America) ⇒ Hostile(x)

7 American(West)

8 Enemy(Nono,America)



At step-4, we can infer facts Missile(M1)

Backward chaining example

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

2 3 Owns(Nono,M_1) ∧ Missile(M_1)

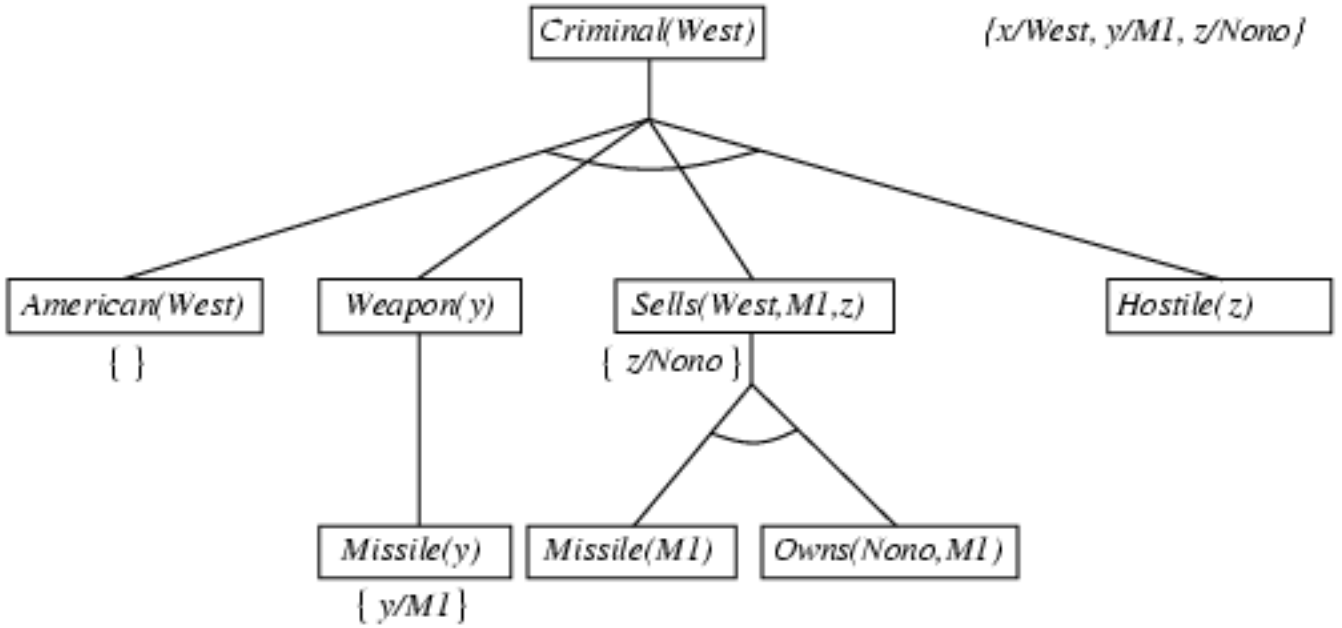
4 Missile(x) ∧ Owns(Nono,x) ⇒ Sells(West,x,Nono)

5 Missile(x) ⇒ Weapon(x)

6 Enemy(x,America) ⇒ Hostile(x)

7 American(West)

8 Enemy(Nono,America)



At step-4, we can infer facts Owns(Nono, M1) form Sells(West, M1, z) which satisfies

the **Rule- 4**, with the substitution of Nono in place of z. So these two statements are proved here.

Backward chaining example

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

2 3 Owns(Nono,M_1) ∧ Missile(M_1)

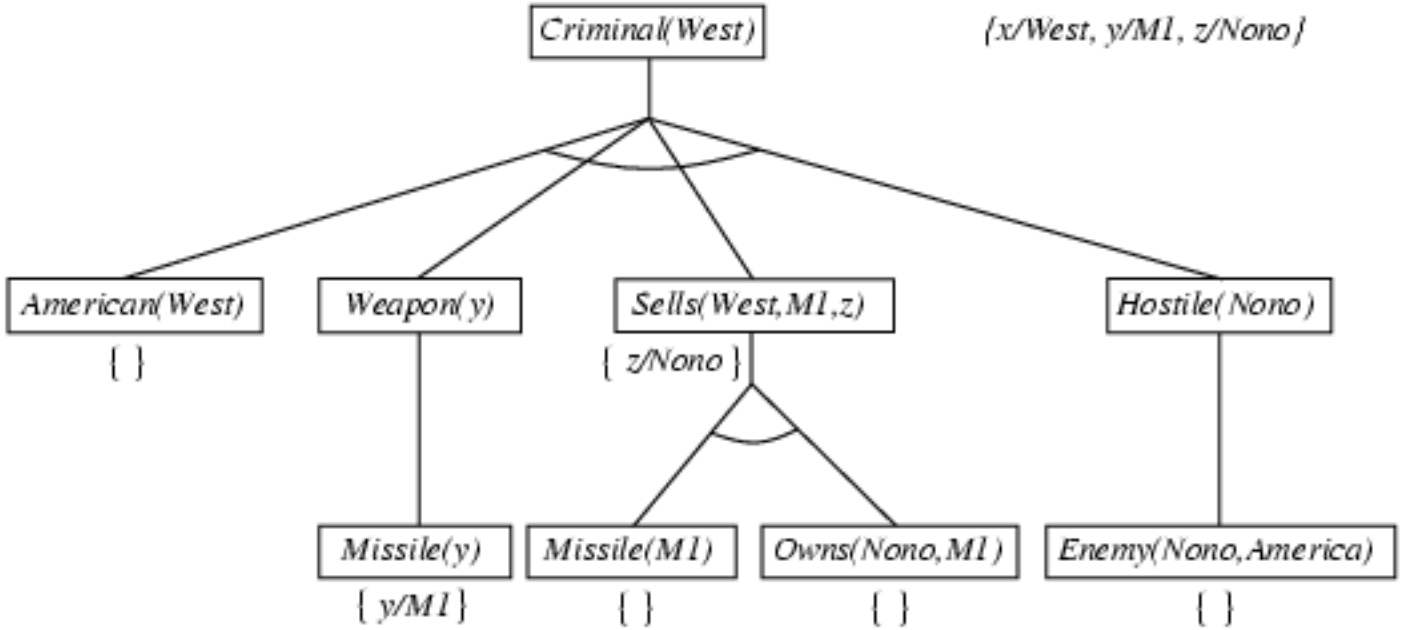
4 Missile(x) ∧ Owns(Nono,x) ⇒ Sells(West,x,Nono)

5 Missile(x) ⇒ Weapon(x)

6 Enemy(x,America) ⇒ Hostile(x)

7 American(West)

8 Enemy(Nono,America)



At step-5, we can infer the fact **Enemy(Nono, America)** from **Hostile(Nono)** which satisfies Rule- 6.

And hence all the statements are proved true using backward chaining.

Backward chaining algorithm

function FOL-BC-ASK(KB, $goals$, θ) **returns** a set of substitutions

inputs: KB, a knowledge base

$goals$, a list of conjuncts forming a query (θ already applied)

θ, the current substitution, initially the empty substitution $\{\}$

local variables: $answers$, a set of substitutions, initially empty

if $goals$ is empty **then return** $\{\theta\}$

$q' \leftarrow$ SUBST(θ, FIRST($goals$))

for each sentence r **in** KB

where STANDARDIZE-APART(r) $= (p_1 \wedge \ldots \wedge p_n \Rightarrow q)$

and $\theta' \leftarrow$ UNIFY(q, q') succeeds

$new\_goals \leftarrow [p_1, \ldots, p_n | \text{REST}(goals)]$

$answers \leftarrow$ FOL-BC-ASK(KB, new_goals, COMPOSE(θ', θ)) $\cup$ $answers$

return $answers$

No.	Forward Chaining	Backward Chaining
1.	Forward chaining starts from known facts and applies inference rule to extract more data unit it reaches to the goal.	Backward chaining starts from the goal and works backward through inference rules to find the required facts that support the goal.
2.	It is a bottom-up approach	It is a top-down approach
3.	Forward chaining is known as data-driven inference technique as we reach to the goal using the available data.	Backward chaining is known as goal-driven technique as we start from the goal and divide into sub-goal to extract the facts.
4.	Forward chaining reasoning applies a breadth-first search strategy.	Backward chaining reasoning applies a depth-first search strategy.

5.	Forward chaining tests for all the available rules	Backward chaining only tests for few required rules.
6.	Forward chaining is suitable for the planning, monitoring, control, and interpretation application.	Backward chaining is suitable for diagnostic, prescription, and debugging application.
7.	Forward chaining can generate an infinite number of possible conclusions.	Backward chaining generates a finite number of possible conclusions.
9.	Forward chaining is aimed for any conclusion.	Backward chaining is only aimed for the required data.

Horn Clause and Definite clause:

- Horn clause and definite clause are the forms of sentences, which enables knowledge base to use a more restricted and efficient inference algorithm.
- Logical inference algorithms use forward and backward chaining approaches, which require KB in the form of the **first-order definite clause**.
- **Definite clause:** A clause which is a disjunction of literals with **exactly one positive literal** is known as a definite clause or strict horn clause.
- **Horn clause:** A clause which is a disjunction of literals with **at most one positive literal** is known as horn clause. Hence all the definite clauses are horn clauses.
- **Example: (¬ p V ¬ q V k)**. It has only one positive literal k. It is equivalent to $p \wedge q \rightarrow k$.

CNF

1. Eliminate biconditionals and implications:
   - Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.
   - Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$.
2. Move $\neg$ inwards:
   - $\neg(\forall x \;\; p) \equiv \exists x \;\; \neg p$,
   - $\neg(\exists x \;\; p) \equiv \forall x \;\; \neg p$,
   - $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$,
   - $\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$,
   - $\neg\neg\alpha \equiv \alpha$.
3. Standardize variables apart by renaming them: each quantifier should use a different variable.
4. Skolemize: each existential variable is replaced by a *Skolem constant* or *Skolem function* of the enclosing universally quantified variables.
   - For instance, $\exists x \, Rich(x)$ becomes $Rich(G1)$ where $G1$ is a new Skolem constant.
   - "Everyone has a heart" $\forall x \;\; Person(x) \Rightarrow \exists y \;\; Heart(y) \wedge Has(x, y)$ becomes $\forall x \;\; Person(x) \Rightarrow Heart(H(x)) \wedge Has(x, H(x))$, where H is a new symbol (Skolem function).
5. Drop universal quantifiers
   - For instance, $\forall x \;\; Person(x)$ becomes $Person(x)$.
6. Distribute $\wedge$ over $\vee$:
   - $(\alpha \wedge \beta) \vee \gamma \equiv (\alpha \vee \gamma) \wedge (\beta \vee \gamma)$.

Conjunctive normal form for first-order logic

- Def: a conjunction of clauses, where each clause is a disjunction of literals.
- Literals can contain variables, which are assumed to be universally quantified.
- *Every sentence of first-order logic can be converted into an inferentially equivalent CNF sentence.*
- Ex:

$$\forall x \;\; American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$$

becomes, in CNF,

$$\neg American(x) \vee \neg Weapon(y) \vee \neg Sells(x, y, z) \vee \neg Hostile(z) \vee Criminal(x).$$

Procedure for conversion to CNF

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

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

The steps are as follows:

♢ Eliminate implications:

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

♢ Move ¬ inwards: In addition to the usual rules for negated connectives, we need rules for negated quantifiers. Thus, we have

$\neg\forall x\ p$ becomes $\exists x\ \neg p$

$\neg\exists x\ p$ becomes $\forall x\ \neg p$.

Our sentence goes through the following transformations:

$$\forall x\ [\exists y\ \neg(\neg Animal(y) \vee Loves(x,y))] \vee [\exists y\ Loves(y,x)].$$
$$\forall x\ [\exists y\ \neg\neg Animal(y) \wedge \neg Loves(x,y)] \vee [\exists y\ Loves(y,x)].$$
$$\forall x\ [\exists y\ Animal(y) \wedge \neg Loves(x,y)] \vee [\exists y\ Loves(y,x)].$$

"Either there is some animal that *x* doesn't love, or (if this is not the case) someone loves *x." Clearly, the meaning of the* original sentence has been preserved.

- Standardize variables: For sentences like $(\forall x \quad P(x)) \vee (\exists x \quad Q(x))$ which use the same variable name twice, change the name of one of the variables. This avoids confusion later when we drop the quantifiers. Thus, we have

$$\forall x \; [\exists y \; Animal(y) \wedge \neg Loves(x, y)] \vee [\exists z \; Loves(z, x)] .$$

- Skolemize: Skolemization is the process of removing existential quantifiers by elimination. In the simple case, it is just like the Existential Instantiation rule of Section 9.1: translate $\exists x \; P(x)$ into $P(A)$, where A is a new constant. If we apply this rule to our sample sentence, however, we obtain

$$\forall x \; [Animal(A) \wedge \neg Loves(x, A)] \vee Loves(B, x)$$

which has the wrong meaning entirely: it says that everyone either fails to love a particular animal A or is loved by some particular entity B. In fact, our original sentence allows each person to fail to love a different animal or to be loved by a different person. Thus, we want the Skolem entities to depend on x:

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

Here F and G are Skolem functions. The general rule is that the arguments of the

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

Here F and G are Skolem functions. The general rule is that the arguments of the Skolem function are all the universally quantified variables in whose scope the existential quantifier appears. As with Existential Instantiation, the Skolemized sentence is satisfiable exactly when the original sentence is satisfiable.

- Drop universal quantifiers: At this point, all remaining variables must be universally quantified. Moreover, the sentence is equivalent to one in which all the universal quantifiers have been moved to the left. We can therefore drop the universal quantifiers:

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

- Distribute $\vee$ over $\wedge$:

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

This step may also require flattening out nested conjunctions and disjunctions.

The sentence is now in CNF and consists of two clauses. It is quite unreadable. (It may help to explain that the Skolem function $F(x)$ refers to the animal potentially unloved by x, whereas $G(x)$ refers to someone who might love x.) Fortunately, humans seldom need look at CNF sentences—the translation process is easily automated.

Resolution in FOL

- **Resolution** is a valid inference rule producing a new clause implied by two clauses containing *complementary literals*
  - A literal is an atomic symbol or its negation, i.e., P, ~P

Resolution

- A KB is actually a set of sentences all of which are true, i.e., a conjunction of sentences.
- To use resolution, put KB into *conjunctive normal form* (CNF), where each sentence written as a disjunction of (one or more) literals

- **Resolution** is a valid inference rule producing a new clause implied by two clauses containing *complementary literals*
  - A literal is an atomic symbol or its negation, i.e., P, ~P

Example

- KB: [P→Q , Q→R∧S]
- KB in CNF: [~P∨Q , ~Q∨R , ~Q∨S]
- Resolve KB(1) and KB(2) producing: ~P∨R *(i.e., P→R)*
- Resolve KB(1) and KB(3) producing: ~P∨S *(i.e., P→S)*
- New KB: [~P∨Q , ~Q∨~R∨~S , ~P∨R , ~P∨S]

Resolution covers many cases

- Modes Ponens
  - from P and $P \rightarrow Q$ derive Q
  - from P and $\neg P \vee Q$ derive Q
- Chaining
  - from $P \rightarrow Q$ and $Q \rightarrow R$ derive $P \rightarrow R$
  - from $(\neg P \vee Q)$ and $(\neg Q \vee R)$ derive $\neg P \vee R$
- Contradiction detection
  - from P and $\neg P$ derive false
  - from P and $\neg P$ derive the empty clause (=false)

Resolution in first-order logic

- Given sentences in *conjunctive normal form:*
  - $P_1 \vee \ldots \vee P_n$ and $Q_1 \vee \ldots \vee Q_m$
  - P_i and Q_i are literals, i.e., positive or negated predicate symbol with its terms

- Example
  - from clause **P(x, f(a)) ∨ P(x, f(y)) ∨ Q(y)**
  - and clause **¬P(z, f(a)) ∨ ¬Q(z)**
  - derive resolvent **P(z, f(y)) ∨ Q(y) ∨ ¬Q(z**
  - Using **θ = {x/z}**

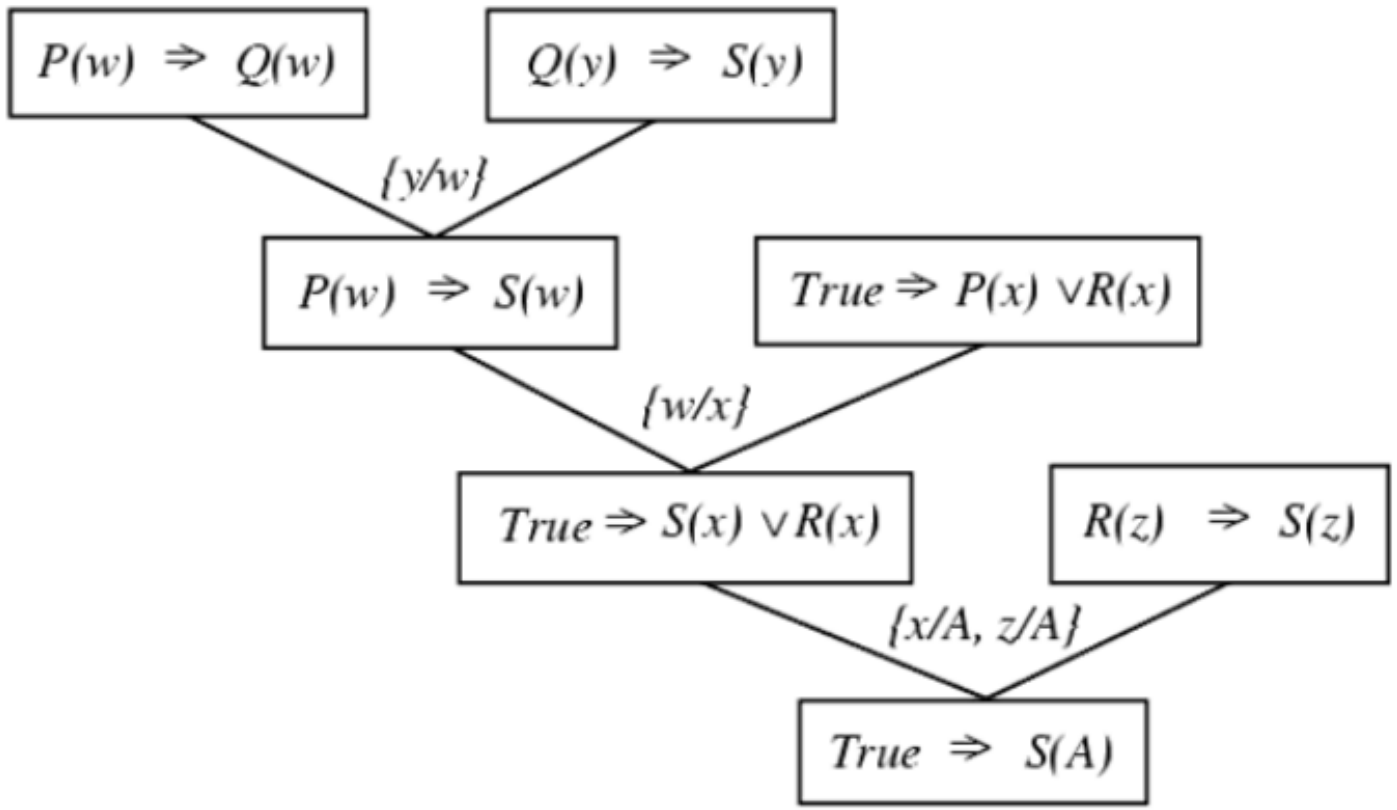
- Example
  - from clause **P(x, f(a)) ∨ P(x, f(y)) ∨ Q(y)**
  - and clause **¬P(z, f(a)) ∨ ¬Q(z)**
  - derive resolvent **P(z, f(y)) ∨ Q(y) ∨ ¬Q(z**
  - Using **θ = {x/z}**

Resolution refutation proofs involve the following steps:

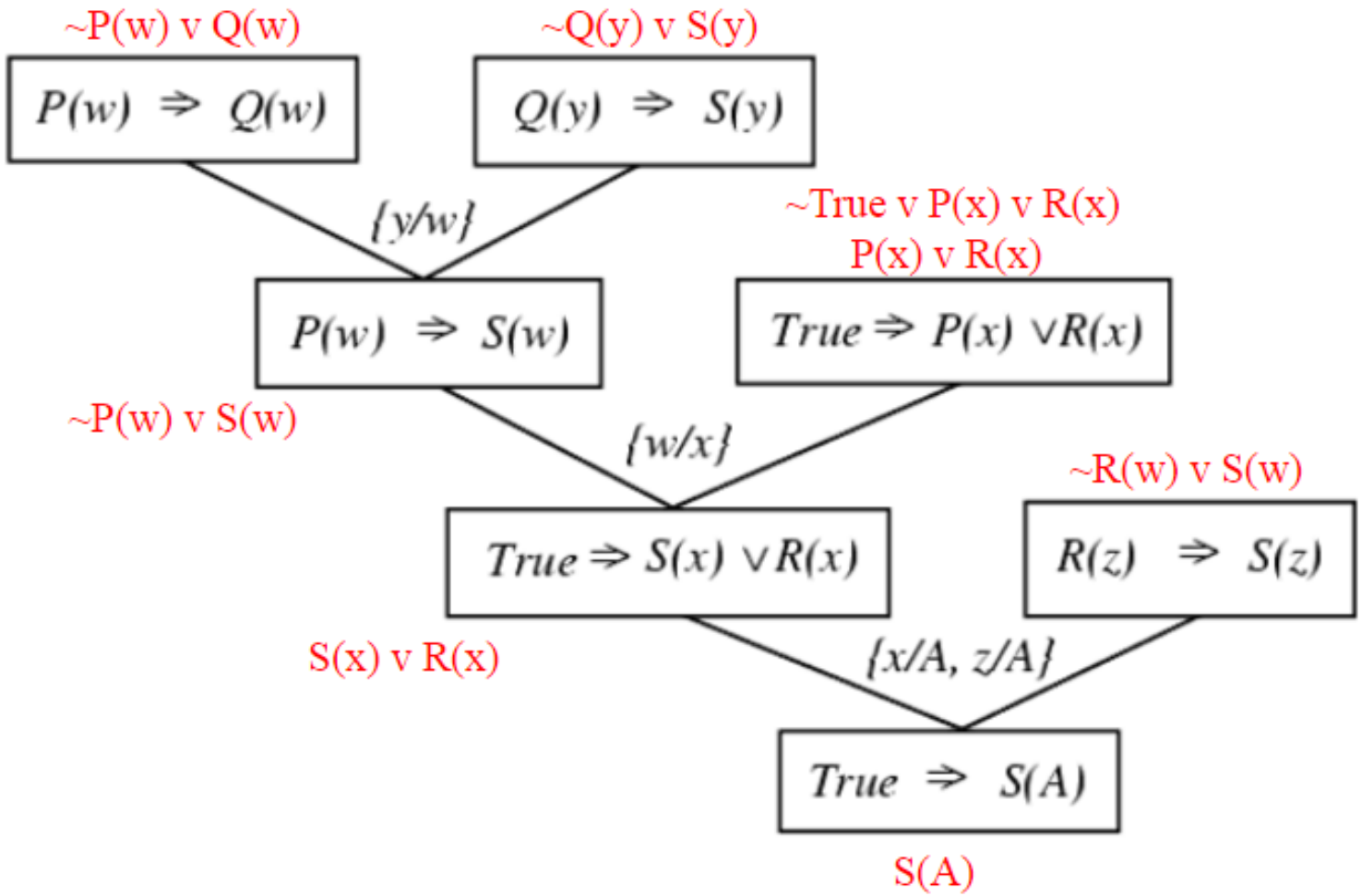
1. Put the premises or axioms into *clause form* .
2. Add the negation of what is to be proved, in clause form, to the set of axioms.
3. *Resolve* these clauses together, producing new clauses that logically follow from them.
4. Produce a contradiction by generating the empty clause.
5. The substitutions used to produce the empty clause are those under which the opposite of the negated goal is true.

The following example illustrates the use of resolution theorem for reasoning with propositional logic.

A resolution proof tree



A resolution proof tree



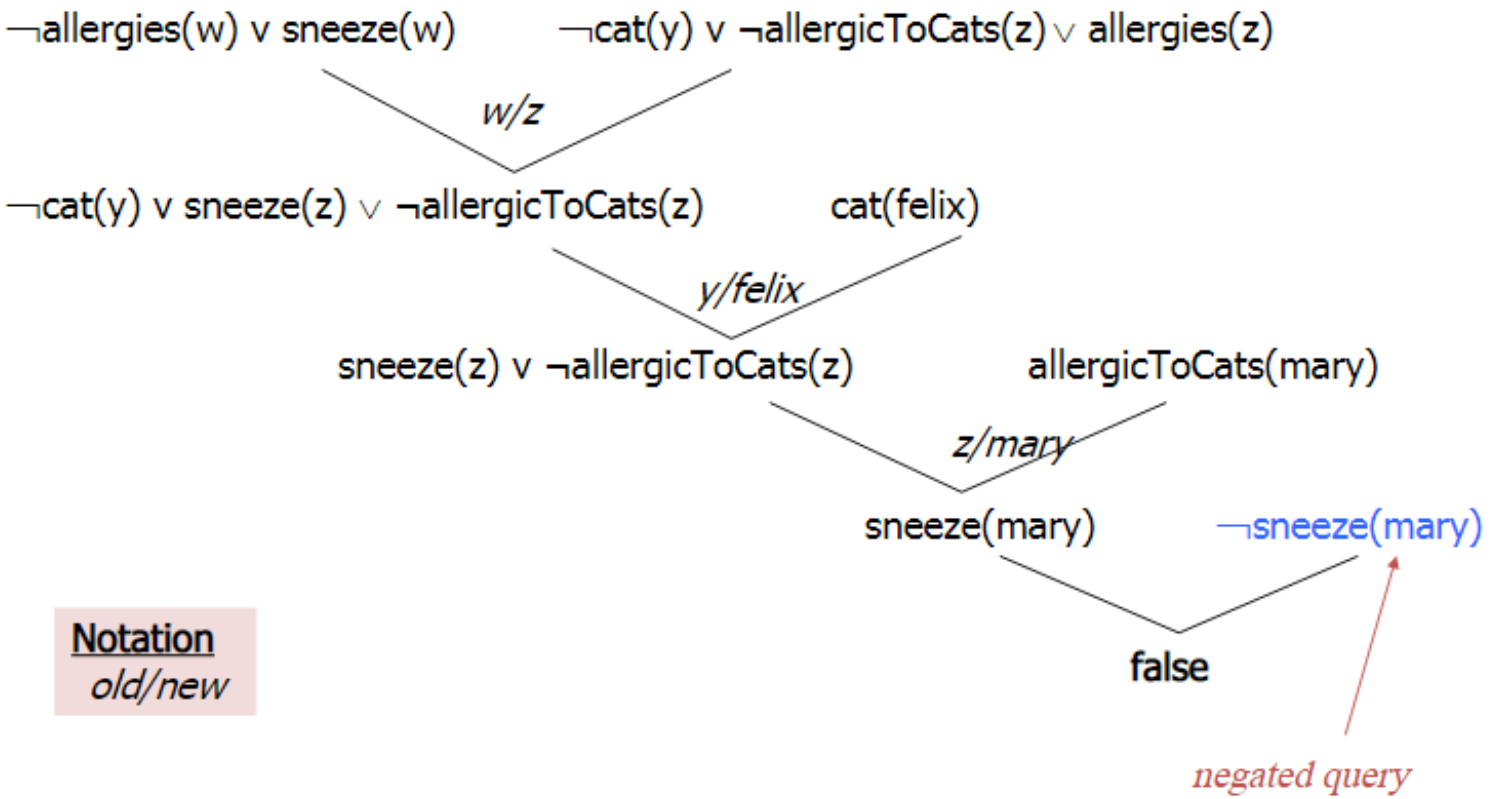
Resolution refutation

- Given a consistent set of axioms KB and goal sentence Q, show that KB |= Q
- **Proof by contradiction:** Add ¬Q to KB and try to prove false, i.e.:
  (KB |- Q) ↔ (KB ∧ ¬Q |- False)
- Resolution is **refutation complete** : it can establish that a given sentence Q is entailed by KB, but can't (in general) generate all logical consequences of a set of sentences
- Also, it cannot be used to prove that Q is **not entailed** by KB
- Resolution **won't always give an answer** since entailment is only semi-decidable
  - And you can't just run two proofs in parallel, one trying to prove Q and the other trying to prove ¬Q, since KB might not entail either one

Resolution example

- KB:
  - allergies(X) → sneeze(X)
  - cat(Y) ∧ allergicToCats(X) → allergies(X)
  - cat(felix)
  - allergicToCats(mary)
- Goal:
  - sneeze(mary)

Refutation resolution proof tree

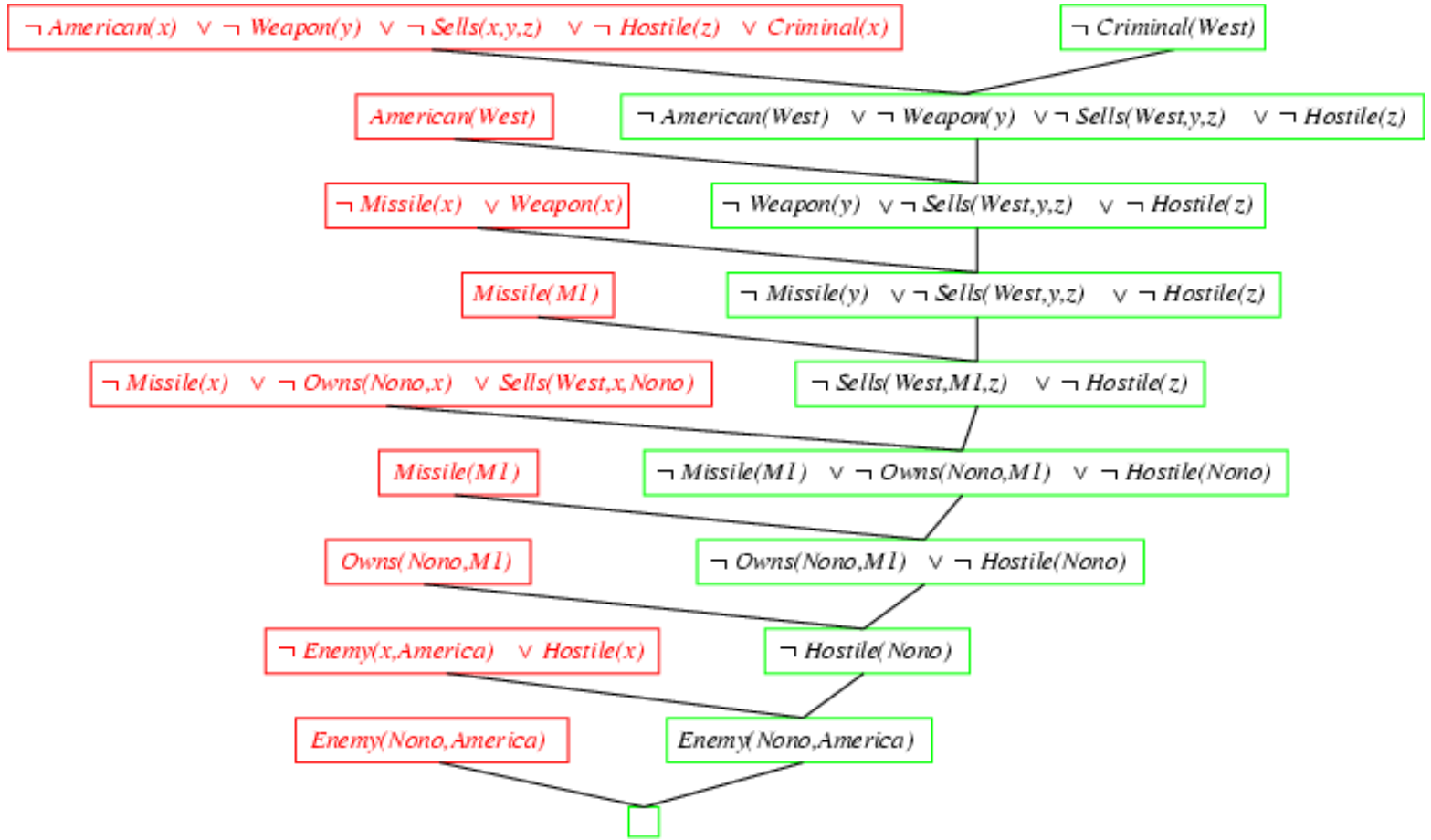


.Exampleproofs

Resolution proves that KB $\models \alpha$ by proving $KB \wedge \neg\alpha$ unsatisfiable, i.e., by deriving the empty clause. The algorithmic approach is identical to the propositional case, described in

: crime example from Section 9.3. The sentences in CNF are

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


¬ 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)

Procedural versus Declarative Knowledge

- consider a set of rules that represent, Knowledge about relationships in the world and Knowledge about how to solve the problem using the content of the rules.
- A representation in which the control information that is necessary to use the knowledge is embedded in the knowledge itself for e.g. computer programs, directions, and recipes; these indicate specific use or implementation;
- The real difference between declarative and procedural views of knowledge lies in where control information reside.

- For example, consider the following
- Man (Marcus)
  Man (Caesar)
  Person (Cleopatra)
  ∀x: Man(x) → Person(x)
  Now, try to answer the question. ?Person(y)
  The knowledge base justifies any of the following answers.

  Y=Marcus

  Y=Caesar

  Y=Cleopatra

- We get more than one value that satisfies the predicate.
- If only one value needed, then the answer to the question will depend on the order in which the assertions examined during the search for a response.
- If the assertions declarative then they do not themselves say anything about how they will be examined. In case of procedural representation, they say how they will examine.

Declarative Knowledge

- A statement in which knowledge specified, but the use to which that knowledge is to be put is not given.
- For example, laws, people's name; these are the facts which can stand alone, not dependent on other knowledge;
- So to use declarative representation, we must have a program that explains what is to do with the knowledge and how.
- For example, a set of logical assertions can combine with a resolution theorem prover to give a complete program for solving problems but in some cases, the logical assertions can view as a program rather than data to a program.
- Hence the implication statements define the legitimate reasoning paths and automatic assertions provide the starting points of those paths.
- These paths define the execution paths which is similar to the 'if then else "in traditional programming.
- So logical assertions can view as a procedural representation of knowledge.

Matching

- **FC and BC are applying appropriate rules to the individual states. Clever searching involves choosing among the rules that can be applied at a particular point. How to take out from the whole collection of rules that can be applied at a specified point?**
- **Matching among current state and the precondition of the rules is discussed as below.**

- Indexing:
- One way is to simple search through all rules. Comparing each one's preconditions to the current state and extracting all the ones that match. But there are 2 problems:
  - I will be necessary to large number of rules to solve interesting problems, which is inefficient.
  - It is not always clear immediately, whether rule's preconditions are satisfied by a particular state.
- Indexing can be handled easily by considering the starting node for the first matching instead of searching through the rules. For example in a board game we need to assign index to each board position. All the rules describing the board position will be stored under the same key.
- Precondition of the rules match exact position of board configuration. But generalization is difficult. But if generalization is done, simple indexing is not possible. There is trade off between ease of writing rules(when increased by high level description) and the simplicity of the matching process(which is decreased by such description)