KNOWLEDGE REPRESENTATION

Representations and Mappings

- In order to solve complex problems encountered in artificial intelligence, one needs both a large amount of knowledge and some mechanism for manipulating that knowledge to create solutions.

- Knowledge and Representation are two distinct entities. They play central but distinguishable roles in the intelligent system.

- Knowledge is a description of the world. It determines a system's competence by what it knows.

- Moreover, Representation is the way knowledge is encoded. It defines a system's performance in doing something.

- Different types of knowledge require different kinds of representation.

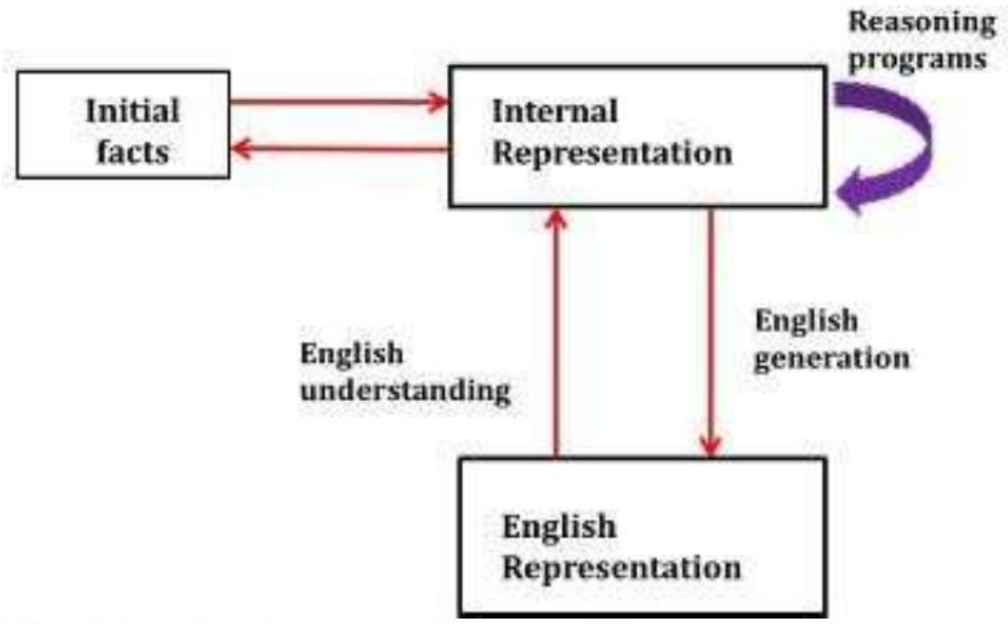


Fig: Mapping between Facts and Representations

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“
    - Exists within a human being;
    - It is embodied.
    - Difficult to articulate formally.
    - Difficult to communicate or share.
    - Moreover, Hard to steal or copy.
    - Drawn from experience, action, subjective insight
2. Explicit formal type of knowledge, Explicit
    - Explicit knowledge
    - Exists outside a human being;
    - It is embedded.
    - Can be articulated formally.
    - Also, Can be shared, copied, processed and stored.
    - So, Easy to steal or copy
    - Drawn from the artifact of some type as a principle, procedure, process, concepts. A variety of ways of representing knowledge have been exploited in AI programs.

There are two different kinds of entities, we are dealing with.

1. Facts: Truth in some relevant world. Things we want to represent.
2. Also, Representation of facts in some chosen formalism. Things we will actually be able to manipulate.

These entities structured at two levels:

1. The knowledge level, at which facts described.
2. Moreover, The symbol level, at which representation of objects defined in terms of symbols that can manipulate by programs

Framework of Knowledge Representation

- The computer requires a well-defined problem description to process and provide a well-defined acceptable solution.

- Moreover, To collect fragments of knowledge we need first to formulate a description in our spoken language and then represent it in formal language so that computer can understand.

- Also, The computer can then use an algorithm to compute an answer. So, This process illustrated as,

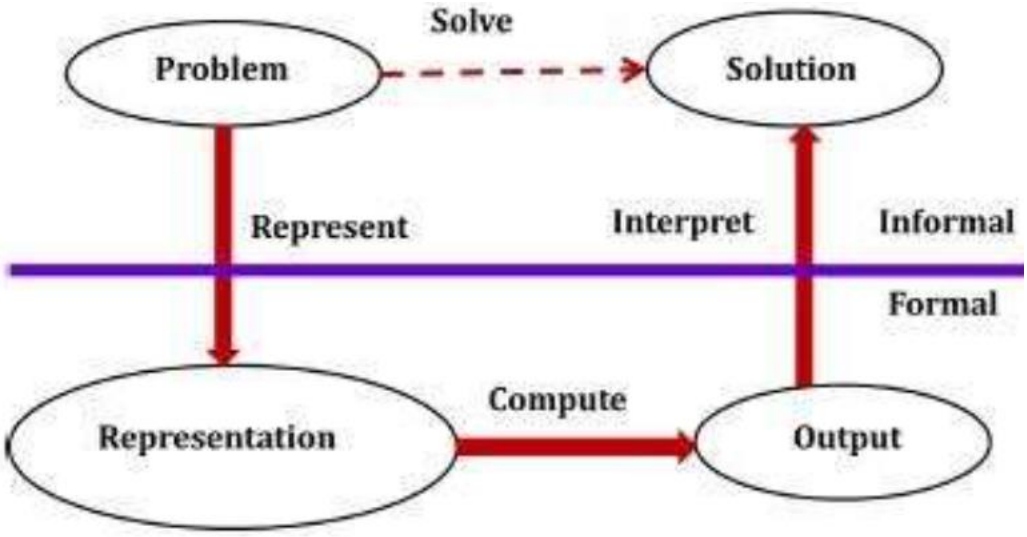


Fig: Knowledge Representation Framework

1. The steps are:
    - The informal formalism of the problem takes place first.
    - It then represented formally and the computer produces an output.
    - This output can then represented in an informally described solution that user understands or checks for consistency. The Problem solving requires,
    - Formal knowledge representation, and
    - Moreover, Conversion of informal knowledge to a formal knowledge that is the conversion of implicit knowledge to explicit knowledge.

Mapping between Facts and Representation

- Knowledge is a collection of facts from some domain.
- Also, We need a representation of "facts" that can manipulate by a program.
- Moreover, Normal English is insufficient, too hard currently for a computer program to draw inferences in natural languages.
- Thus some symbolic representation is necessary.

A good knowledge representation enables fast and accurate access to knowledge and understanding of the content.
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.

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

Knowledge Representation Schemes Relational Knowledge

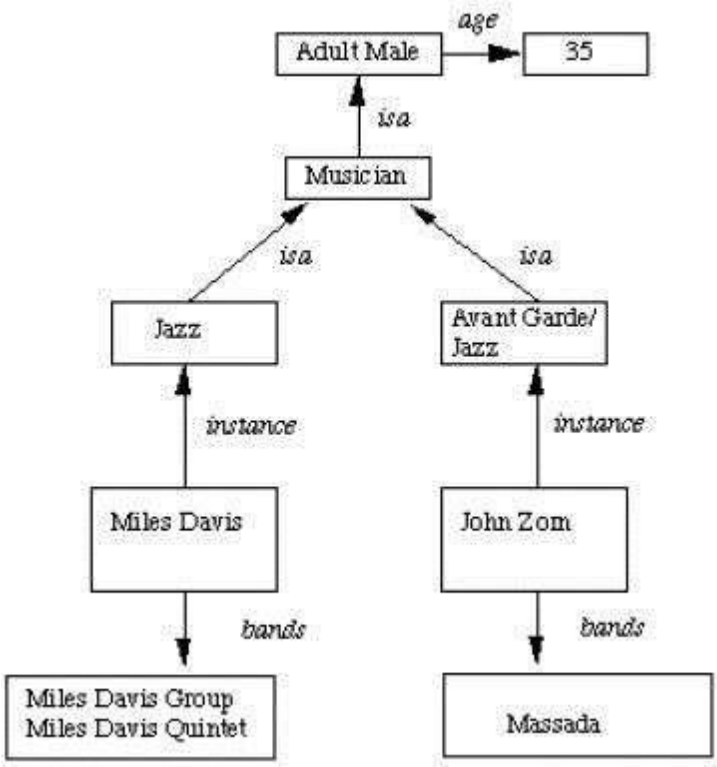
- The simplest way to represent declarative facts is a set of relations of the same sort used in the database system.
- Provides a framework to compare two objects based on equivalent attributes.
  - Any instance in which two different objects are compared is a relational type of knowledge.
- The table below shows a simple way to store facts.
  - Also, The facts about a set of objects are put systematically in columns.
  - This representation provides little opportunity for inference.

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

- Given the facts, it is not possible to answer a simple question such as: “Who is the heaviest player?”
- Also, But 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

- Here the knowledge elements inherit attributes from their parents.
- The knowledge embodied in the design hierarchies found in the functional, physical and process domains.
- Within the hierarchy, elements inherit attributes from their parents, but in many cases, not all attributes of the parent elements prescribed to the child elements.

- Also, 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.



Boxed nodes — objects and values of attributes of objects.

- 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.

- • Example: given a set of relations and values, one may infer other values or relations. Advantages

  predicate logic (a mathematical deduction) used to infer from a set of attributes. Moreover, Inference through predicate logic uses a set of logical operations to relate individual data.

- • Represent knowledge as formal logic: All dogs have tails ∀x: dog(x) → hastail(x)
- • Advantages:
  - • A set of strict rules.
  - • Can use to derive more facts.
  - • Also, Truths of new statements can be verified.
  - • Guaranteed correctness.
- • So, Many inference procedures available to implement standard rules of logic popular in AI systems. e.g Automated theorem proving.

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;
- • Moreover, Knowledge encoded in some procedures, small programs that know how to do specific things, how to proceed.
- • Advantages:
  - • Heuristic or domain-specific knowledge can represent.
  - • Moreover, Extended logical inferences, such as default reasoning facilitated.
  - • Also, Side effects of actions may model. Some rules may become false in time.

    Keeping track of this in large systems may be tricky.

- • Disadvantages:
  - • Completeness — not all cases may represent.
  - • Consistency — not all deductions may be correct. e.g If we know that Fred is a

    bird we might deduce that Fred can fly. Later we might discover that Fred is an emu.

  - • Modularity sacrificed. Changes in knowledge base might have far-reaching effects.
  - • Cumbersome control information

Knowledge Representation and Reasoning

Intelligent agents should have capacity for:

- **Perceiving**, that is, acquiring information from environment,

- **Knowledge Representation**, that is, representing its understanding of the world,
- **Reasoning**, that is, inferring the implications of what it knows and of the choices it has, and
- **Acting**, that is, choosing what it want to do and carry it out.

Representation of knowledge and the reasoning process are 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. Each sentence is 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. Specifically, we will be dealing with formal logic. The advantage of using formal logic as a language of AI is that it is precise and definite. This allows programs to be written which are declarative - they describe what is true and not how to solve problems. This also allows for automated reasoning techniques for general purpose inferencing.

This, however, leads to some severe limitations. Clearly, a large portion of the reasoning carried out by humans depends on handling knowledge that is uncertain. Logic cannot represent this uncertainty well. Similarly, natural language reasoning requires inferring hidden state, namely, the intention of the speaker. When we say, "One of the wheel of the car is flat.", we know that it has three wheels left. Humans can cope with virtually infinite variety of utterances using a finite store of commonsense knowledge. Formal logic has difficulty with this kind of ambiguity.

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

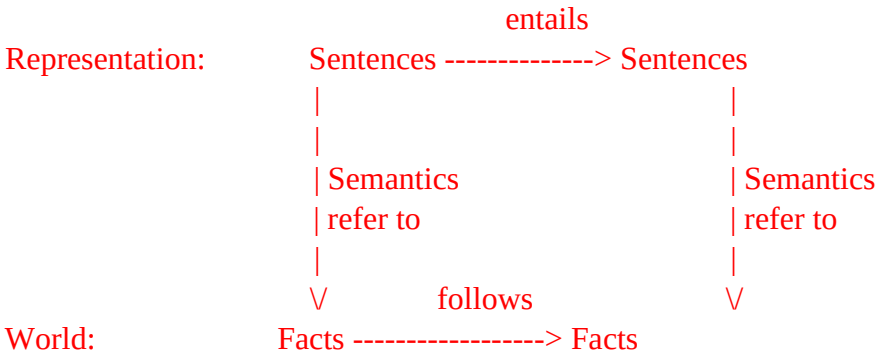
Figure 8.1 Formal languages and their ontological and epistemological commitments.

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. Hence, also specifies 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.

Facts are claims about the world that are True or False, whereas a **representation** is an expression (sentence) in some language that can be encoded in a computer program and stands for the objects and relations in the world. We need to ensure that the representation is consistent with reality, so that the following figure holds:



There are a number of logical systems with different syntax and semantics. We list below a few of them.

- Propositional logic

  All objects described are fixed or unique

  "John is a student" student(john)

  Here John refers to one unique person.

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" o
- 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)

Example of a formula : ((((a ∧ ¬b) ∨ c → d) ↔ ¬ (a ∨ 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.

- Propositional logic is the simplest logic – illustrates basic ideas
- The proposition symbols P_1, P_2 etc are sentences
  - If S is a sentence, ¬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)

◇ 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

- 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.")
- Given the truth values of all of the constituent symbols in a sentence, that sentence can be "evaluated" to determine its truth value (True or False). This is called an **interpretation** of the sentence.

- A **model** is an interpretation (i.e., an assignment of truth values to symbols) of a set of sentences such that each sentence is True. A model is just a formal mathematical structure that "stands in" for the world.

- A **valid** sentence (also called a **tautology**) is a sentence that is True under *all* interpretations. Hence, no matter what the world is actually like or what the semantics is, the sentence is True. For example "It's raining or it's not raining."

- An **inconsistent** sentence (also called **unsatisfiable** or a **contradiction**) is a sentence that is False under *all* interpretations. Hence the world is never like what it describes. For example, "It's raining and it's not raining."

- 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

Entailment ($\models$): Given 2 *sentences* p and q we say p *entails* q, written $p \models q$, if q holds in every model that p holds.

Example: Entailment

$$p \wedge (p \Rightarrow q) \models q$$

Show that:

$$p \wedge (p \Rightarrow q)$$

Proof: For any model M in which holds then we know that *p* holds in M and holds in M. Since $p \Rightarrow q$ *p* $p \Rightarrow q$ holds in M then since holds in M, *q* must hold in

$$p \wedge (p \Rightarrow q) \qquad p \wedge (p \Rightarrow q) \models q$$

M. Therefore *q* holds in every model that holds and so .

As we have noted models affect equivalence and so we repeat the definition again and give an example of a proof of equivalence.

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* A *WumpusAlive*) $\Rightarrow$ *Shoot* and (*WumpusAhead* A *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* A *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.

Rules of Inference

Here are some examples of sound rules of inference. Each can be shown to be sound once and for all using a truth table. The left column contains the premise sentence(s), and the right column contains the derived sentence. We write each of these derivations as A |- B , where A is the premise and B is the derived sentence.

Name	**Premise(s)**	**Derived Sentence**
Modus Ponens	A, A => B	B
And Introduction	A, B	A ^ B
And Elimination	A ^ B	A
Double Negation	~~A	A
Unit Resolution	A v B, ~B	A
Resolution	A v B, ~B v C	A v C

In addition to the above rules of inference one also requires a set of equivalences of propositional logic like “A /\ B” is equivalent to “B /\ A”. A number of such equivalences were presented in the discussion on propositional logic.

Inference ($\vdash$): Given 2 *sentences* p and q we say q *is inferred from* p, written $p \vdash q$, if there is a sequence of rules of inference that apply to p and allow q to be added.

Notice that inference is not directly related to truth; i.e. we can infer a sentence provided we have rules of inference that produce the sentence from the original sentences.

However, if rules of inference are to be useful we wish them to be related to entailment. Ideally we would like:

$p \vdash q$ $p \models q$ iff but this equivalence may fail in two ways:

- $p \vdash q$ $p \not\models q$ but

We have inferred ***q*** by applying rules of inference to ***p***, but there is some model in which p holds but q does not hold. In this case the rules of inference have inferred ``too much''.

- $p \models q \quad p \not\vdash q$ but

q is a sentence which holds in all models in which ***p*** holds, but we cannot find rules of inference that will infer ***q*** from ***p***. In this case the rules of inference are insufficient to infer the things we want to be able to infer.

USING PREDICATE LOGIC

Representation of Simple Facts in Logic

1. Propositional logic is useful because it is simple to deal with and a decision procedure for it exists.
   Also, In order to draw conclusions, facts are represented in a more convenient way as,

   1. Marcus is a man.
   • man(Marcus)

   2. Plato is a man.
   • man(Plato)

   3. All men are mortal. • mortal(men)

   But propositional logic fails to capture the relationship between an individual being a man and that individual being a mortal.

   - • How can these sentences be represented so that we can infer the third sentence from the first two?
   - • Also, Propositional logic commits only to the existence of facts that may or may not be the case in the world being represented.
   - • Moreover, It has a simple syntax and simple semantics. It suffices to illustrate the process of inference.
   - • Propositional logic quickly becomes impractical, even for very small worlds. Predicate logic
     First-order Predicate logic (FOPL) 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"

First-order Predicate logic (FOPL) provides

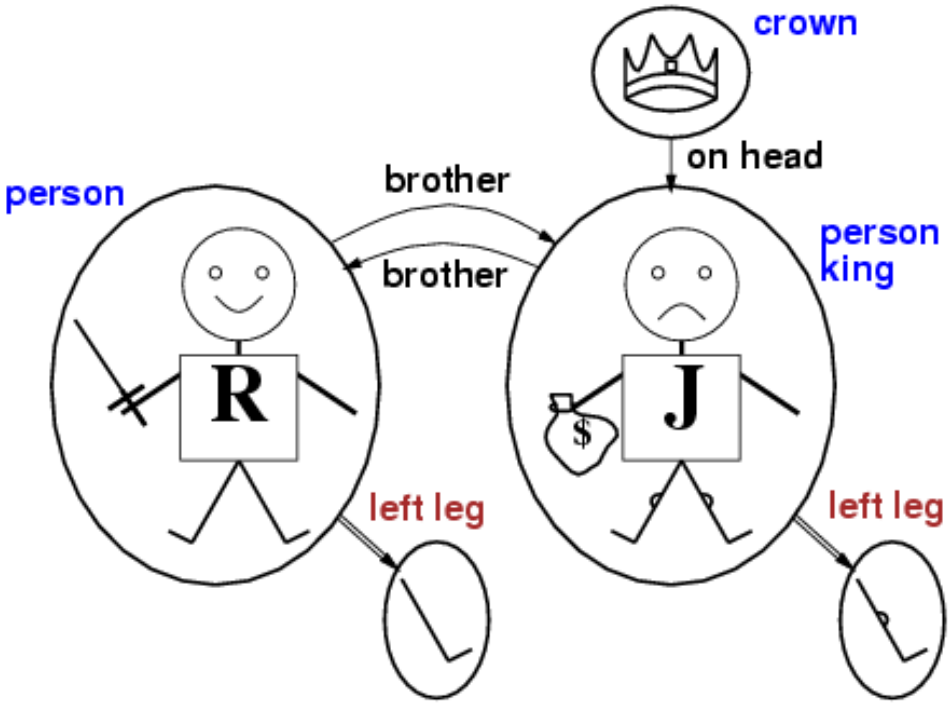
- Constants: a, b, dog33. Name a specific object.
- Variables: X, Y. Refer to an object without naming it.
- Functions: Mapping from objects to objects.
- Terms: Refer to objects
- Atomic Sentences: in(dad-of(X), food6) Can be true or false, Correspond to propositional

  symbols P, Q.
  A well-formed formula (wff) is a sentence containing no "free" variables. So, That is, all variables are "bound" by universal or existential quantifiers.
  $(\forall x)P(x, y)$ has x bound as a universally quantified variable, but y is free.

  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

Example - Model



Symbols are the basic syntactic elements of first-order logic. Symbols stand for objects, relations, and functions.

The symbols are of three kinds:

Constant symbols which stand for objects; Example: John, Richard

Predicate symbols, which stand for relations; Example: OnHead, Person, King, and Crown

Function symbols, which stand for functions. Example: left leg

Symbols will begin with uppercase letters.

Interpretation The semantics must relate sentences to models in order to determine truth. For this to happen, we need an interpretation that specifies exactly which objects, relations and functions are referred to by the constant, predicate, and function symbols.

For Example:

Richard refers to Richard the Lionheart and John refers to the evil king John.

Brother refers to the brotherhood relation

OnHead refers to the "on head relation that holds between the crown and King John; Person, King, and Crown refer to the sets of objects that are persons, kings, and crowns.

LeftLeg refers to the "left leg" function,

The truth of any sentence is determined by a model and an interpretation for the sentence's symbols. Therefore, entailment, validity, and so on are defined in terms of all possible models and all possible interpretations. The number of domain elements in each model may be unbounded-for example, the domain elements may be integers or real numbers.

Term

A term is a logical expression that refers to an object. Constant symbols are therefore terms. Complex Terms A complex term is just a complicated kind of name. A complex term is formed by a function symbol followed by a parenthesized list of terms as arguments to the function symbol For example: "King John's left leg" Instead of using a constant symbol, we use LeftLeg(John). The formal semantics of terms :

Consider a term f (tl,. . . , t,). The function symbol frefers to some function in the model (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 Fapplied to dl, . . . , d,. For example,: the LeftLeg function symbol refers to the function " (King John) -+ John's left leg" and John refers to King John, then LeftLeg(John) refers to King John's left leg. In this way, the interpretation fixes the referent of every term.

Atomic sentences

An atomic sentence is formed from a predicate symbol followed by a parenthesized list of terms: For Example: Brother(Richard, John).

Atomic sentences can have complex terms as arguments. For Example: Married (Father(Richard), Mother(John)).

An atomic sentence is true in a given model, under a given interpretation, if the relation referred to by the predicate symbol holds among the objects referred to by the arguments

Complex sentences Complex sentences can be constructed using logical Connectives, just as in propositional calculus. For Example:

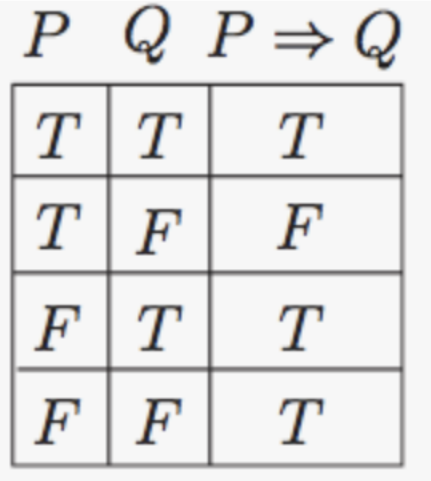
- ✓ $\neg$Brother (LeftLeg(Richard), John)
- ✓ Brother (Richard , John) $\wedge$ Brother (John, Richard)
- ✓ King(Richard) $\vee$ King(John)
- ✓ $\neg$King(Richard) $\Rightarrow$ King(John)

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 $\forall x\ P$, where *P* is any logical expression, says that *P* is true for every object *x.*
- More precisely, $\forall x$ 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.
- This sounds complicated, but it is really just a careful way of stating the intuitive meaning of universal quantification. We can extend the interpretation in five ways:
- x →Richard
- *x* → King John,
- *x* →Richard's left leg,
- x →John's left leg,
- *x* → the crown.
- The universally quantified sentence $\forall x\ King(x) \Rightarrow Person(x)$ is true under the original interpretation if the sentence $King(x) \Rightarrow Person(x)$ is true in each of the five extended interpretations. That is, the universally quantified sentence is equivalent to asserting the following five sentences:
- Richard is a king ⇒Richard the Lionheart is a person.
- King John is a king ⇒King John is a person.
- Richard's left leg is a king ⇒Richard's left leg is a person.
- John's left leg is a king ⇒John's left leg is a person.
- The crown is a king ⇒the crown is a person.



P	Q	$P \Rightarrow Q$
T	T	T
T	F	F
F	T	T
F	F	T

- Thus, by asserting the universally quantified sentence, which is equivalent to asserting a whole list of individual implications, we end up asserting the conclusion of the rule just for those objects for whom the premise is true and saying nothing at all about those individuals for whom the premise is false.
- Thus, the truth-table entries for ⇒turn out to be perfect for writing general rules with universal quantifiers. Now an implication is true if both premise and conclusion are true, or *if* ***its premise is false.***
- A common mistake, made frequently even by diligent readers who have read this paragraph several times, is to use conjunction instead of implication. The sentence
- ∀*x King (x) A Person (x)*

would be equivalent to asserting

- Richard is a king ∧ Richard the Lionheart is a person,
- King John is a king ∧ King John is a person,
- Richard's left leg is a king ∧ Richard's left leg is a person,

and so on. Obviously, this does not capture what we want.

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 $\wedge$ John's left leg is on John's head;
    - The crown is a crown $\wedge$ 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.

- Consider the following sentence:

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

- On the surface, this might look like a reasonable rendition of our sentence. Applying the semantics, we see that the sentence says that at least one of the following assertions is true:
    - Richard the Lionheart is a crown $\Rightarrow$Richard the Lionheart is on John's head;
    - King John is a crown $\Rightarrow$ King John is on John's head;
    - Richard's left leg is a crown $\Rightarrow$ Richard's left leg is on John's head;

and so on. Now an implication is true if both premise and conclusion are true, or *if* ***its premise is false.***

Nested Quantifiers

One can express more complex sentences using multiple quantifiers.

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) $\Leftrightarrow$Sibling(y, x).
In other cases we will have mixtures.

For example: 1. “Everybody loves somebody” means that for every person, there is someone that person loves: $\forall x \exists y$ Loves(x, y) . 2. 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 $\forall$and $\exists$

Universal and Existential quantifiers are actually intimately connected with each other, through negation.

Example assertions: 1. " Everyone dislikes medicine" is the same as asserting " there does not exist someone who likes medicine" , and vice versa: "∀x ¬Likes(x, medicine)" is equivalent to "¬∃x Likes(x, medicine)".

"Everyone likes ice cream" means that " there is no one who does not like ice cream" : ∀xLikes(x, IceCream) is equivalent to ¬∃x ¬Likes(x, IceCream) .

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:

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.

Equality

First-order logic includes one more way to make atomic sentences, other than using a predicateand terms .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.The equality symbol can be used to state facts about a given function.It can also be used with negation to insist that two terms are not the same object.

For example,
“Richard has at least two brothers” can be written as, ∃x, y Brother (x,Richard) ∧Brother (y,Richard)

∧¬(x=y) . The sentence ∃x, y Brother (x,Richard) ∧Brother (y,Richard) does not have the intended meaning. In particular, it is true only in the model where Richard has only one brother considering the extended interpretation in which both x and y are assigned to King John. The addition of ¬(x=y) rules out such models.

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

Well formed Formulas:

Consider the following example that shows the use of predicate logic as a way of representing knowledge.

1. Marcus was a man.
2. Marcus was a Pompeian.
3. All Pompeians were Romans.
4. Caesar was a ruler.
5. Also, All Pompeians were either loyal to Caesar or hated him.
6. Everyone is loyal to someone.
7. People only try to assassinate rulers they are not loyal to.
8. 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.
• ∀x: Pompeian(x) → Roman(x)

4. Caesar was a ruler.
• ruler(Caesar)

5. All Pompeians were either loyal to Caesar or hated him.

- inclusive-or
- ∀x: Roman(x) → loyalto(x, Caesar) ∨ hate(x, Caesar)
- exclusive-or
- ∀x: Roman(x) → (loyalto(x, Caesar) ∧¬ hate(x, Caesar)) ∨
- (¬loyalto(x, Caesar) ∧ hate(x, Caesar))

6. Everyone is loyal to someone. • ∀x: ∃y: loyalto(x, y)
7. People only try to assassinate rulers they are not loyal to.
    0. • ∀x: ∀y: person(x) ∧ ruler(y) ∧ tryassassinate(x, y)
    1. • →¬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: ¬ Ioyalto(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.

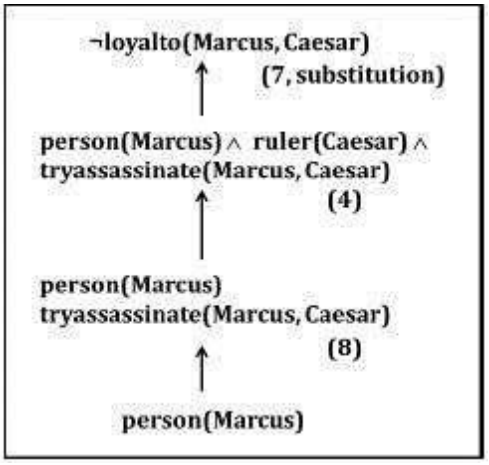


Figure: An attempt to prove ¬loyalto(Marcus, Caesar).

- • The problem is that, although we know that Marcus was a man, we do not have any way to conclude from that that Marcus was a person. Also, We need to add the representation of another fact to our system, namely: ∀ man(x) → person(x)
- • Now we can satisfy the last goal and produce a proof that Marcus was not loyal to Caesar.
- • Moreover, From this simple example, we see that three important issues must be addressed in the process of converting English sentences into logical statements and then using those statements to deduce new ones:
    0. Many English sentences are ambiguous (for example, 5, 6, and 7 above). Choosing the correct interpretation may be difficult.
    1. Also, There is often a choice of how to represent the knowledge. Simple representations are desirable, but they may exclude certain kinds of reasoning.
    2. Similalry, Even in very simple situations, a set of sentences is unlikely to contain all the information necessary to reason about the topic at hand. In order to be able to use a set of statements effectively, It is usually necessary to have access to another set of statements that represent facts that people consider too obvious to mention.

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.

• 4.2 shows the first five sentences of the last section represented in logic in three different ways.

- The first part of the figure contains the representations we have already discussed. In these representations, 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.

- The second part of the figure contains representations that use the instance predicate explicitly.

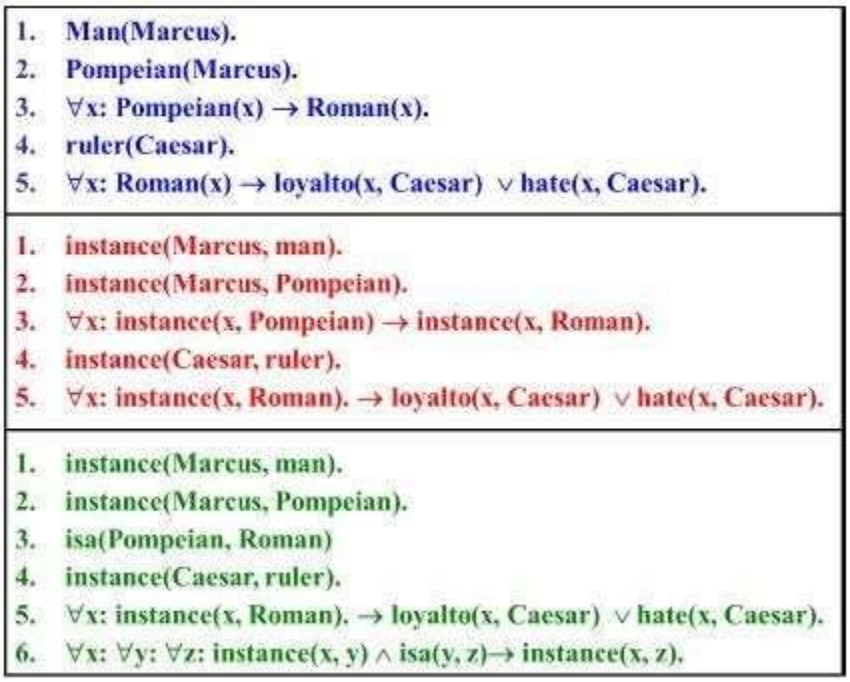


Figure: Three ways of representing class membership: ISA Relationships

- 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.

- 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.

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.

- Also, As soon as we attempt to follow either of those paths rigorously, however, we discover, just as we did in the last example, that we need some additional knowledge. 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.

x: t: [alive(x, t) → ¬ dead(x, t)] [¬ dead(x, t) → alive(x, t)]

9) If someone dies, then he is dead at all later times.

x: t1: t2: 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.

Universal Instantiation (UI):
The rule says that we can infer any sentence obtained by substituting a ground term (a term without variables) for the variable. Let SUBST (θ) 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, there is a sentence in knowledge base stating that all greedy kings are Evils

$$\forall x \ \ King(x) \ A \ Greedy(x) \Rightarrow Evil(x) \ .$$

For the variable x, with the substitutions like {x/John},{x/Richard}the following sentences can be inferred.

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

Thus a universally quantified sentence can be replaced by the set of all possible instantiations.

Existential Instantiation (EI):

The existential sentence says there is some object satisfying a condition, and the instantiation process is just giving a name to that object, that name must not already belong to another object. This new name is called a Skolem constant. Existential Instantiation is a special case of a more general process called “skolemization”.

For any sentence a, variable v, and constant symbol k that does not appear elsewhere in the knowledge base,

$$\frac{\exists v \ \ \alpha}{\text{SUBST}(\{v/k\}, \alpha)}$$

For example, from the sentence

$$\exists x \ \ Crown(x) \ A \ OnHead(x, John)$$

So, we can infer the sentence

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

As long as C_1 does not appear elsewhere in the knowledge base. Thus an existentially quantified sentence can be replaced by one instantiation

Elimination of Universal and Existential quantifiers should give new knowledge base which can be shown to be inferentially equivalent to old in the sense that it is satisfiable exactly when the original knowledge base is satisfiable.

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.**

First Order Inference Rule:

The key advantage of lifted inference rules over propositionalization is that they make only those substitutions which are required to allow particular inferences to proceed.

Generalized Modus Ponens:

If there is some substitution θ that makes the premise of the implication identical to sentences already in the knowledge base, then we can assert the conclusion of the implication, after applying θ. This inference process can be captured as a single inference rule called General ized Modus Ponens which is a lifted version of Modus Ponens-it raises Modus Ponens from propositional to first-order logic

For atomic sentences pi, pi ', and q, where there is a substitution θ such that SUBST(θ , pi) = SUBST(θ , pi '), for all i,

$p1', p2', ..., pn', (p1 \wedge p2 \wedge ... \wedge pn \Rightarrow q)$

SUBST (θ, q)

There are N + 1 premises to this rule, N atomic sentences + one implication.

Applying SUBST (θ, q) yields the conclusion we seek. It is a sound inference rule.

Suppose that instead of knowing Greedy (John) in our example we know that everyone is greedy:

$\forall y$ Greedy(y)

We would conclude that Evil(John).

Applying the substitution {x/John, y / John) to the implication premises King (x) and Greedy (x) and the knowledge base sentences King(John) and Greedy(y) will make them identical. Thus, we can infer the conclusion of the implication.

For our example,

p_1' is $King(John)$ $\quad\quad$ p_1 is $King(x)$
p_2' is $Greedy(y)$ $\quad\quad$ p_2 is $Greedy(x)$
θ is $\{x/John, y/John)$ $\quad\quad$ q is $Evil(x)$
SUBST(θ, q) is $Evil(John)$.

Forward Chaining

First-Order Definite Clauses:

A definite clause either is atomic or is an implication whose antecedent is a conjunction of positive literals and whose consequent is a single positive literal. The following are first-order definite clauses:

Unlike propositional literals, first-order literals can include variables, in which case those variables are assumed to be universally quantified. Consider the following problem;

“The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.” We will represent the facts as first-order definite clauses

". . . 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." The sentence 3, Owns (Nono, M1) A Missile (x) is transformed into two definite clauses by Existential Elimination, introducing a new constant

M1: Owns (Nono, M1) ---------- (2)

Missile (Ml) ---------- (3)

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

Missile (x) A Owns (Nono, x) =>Sells (West, z, Nono) ---------- (4)

We will also need to know that missiles are weapons:

Missile (x) => Weapon (x) ---------- (5)

We must know that an enemy of America counts as "hostile":

Enemy (x, America) =>Hostile(x) ---------- (6)

"West, who is American":

American (West) ---------- (7)

"The country Nono, an enemy of America ":

Enemy (Nono, America) ---------- (8)

A simple forward-chaining algorithm:

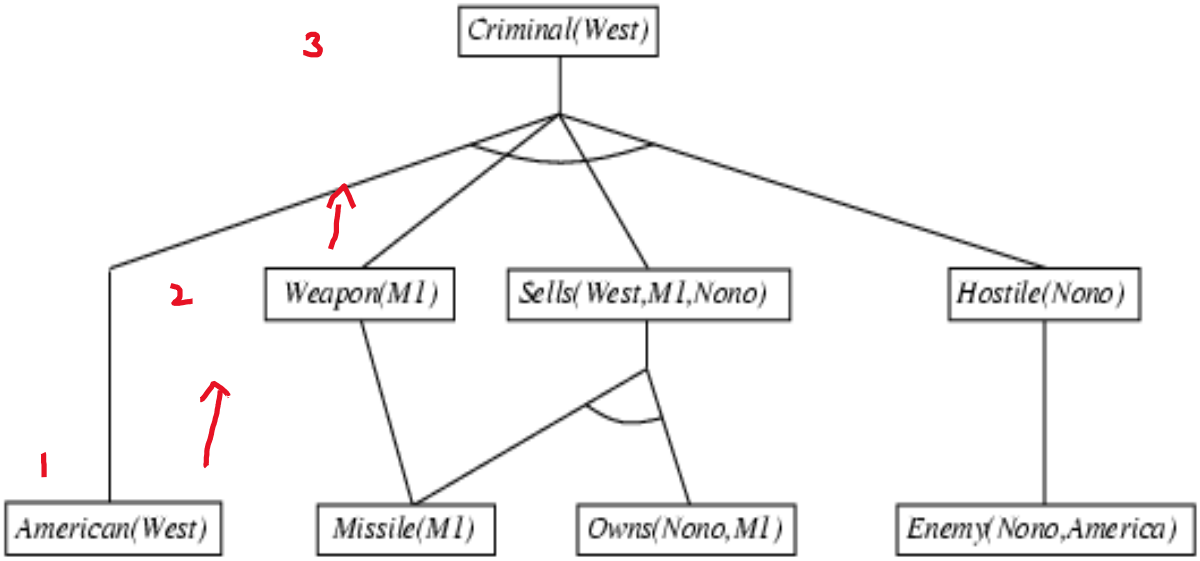
- Starting from the known facts, it triggers all the rules whose premises are satisfied, adding their conclusions of the known facts
- The process repeats until the query is answered or no new facts are added. Notice that a fact is not "new" if it is just renaming of a known fact.

1. 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.

It is sound, because every inference is just an application of Generalized Modus Ponens, it is complete for definite clause knowledge bases; that is, it answers every query whose answers are entailed by any knowledge base of definite clauses .



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

```
function FOL-FC-ASK(KB, α) returns a substitution or false
  repeat until new is empty
    new ← { }
    for each sentence r in KB do
      (p1 ∧ ... ∧ pn ⇒ q) ← STANDARDIZE-APART(r)
      for each θ such that (p1 ∧ ... ∧ pn)θ = (p'1 ∧ ... ∧ p'n)θ
                  for some p'1, ..., p'n in KB
        q' ← SUBST(θ, q)
        if q' is not a renaming of a sentence already in KB or new then do
          add q' to new
          φ ← UNIFY(q', α)
          if φ is not fail then return φ
    add new to KB
  return false
```

4. Backward Chaining

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.

The algorithm takes the first goal in the list and finds every clause in the knowledge base whose head, unifies with the goal. Each such clause creates a new recursive call in which body, of the clause is added to the goal stack.

Remember that facts are clauses with a head but no body, so when a goal unifies with a known fact, no new sub goals are added to the stack and the goal is solved.

The algorithm for backward chaining and proof tree for finding criminal (West) using backward chaining are given below.

Properties of backward chaining:

It is known as a top-down approach.

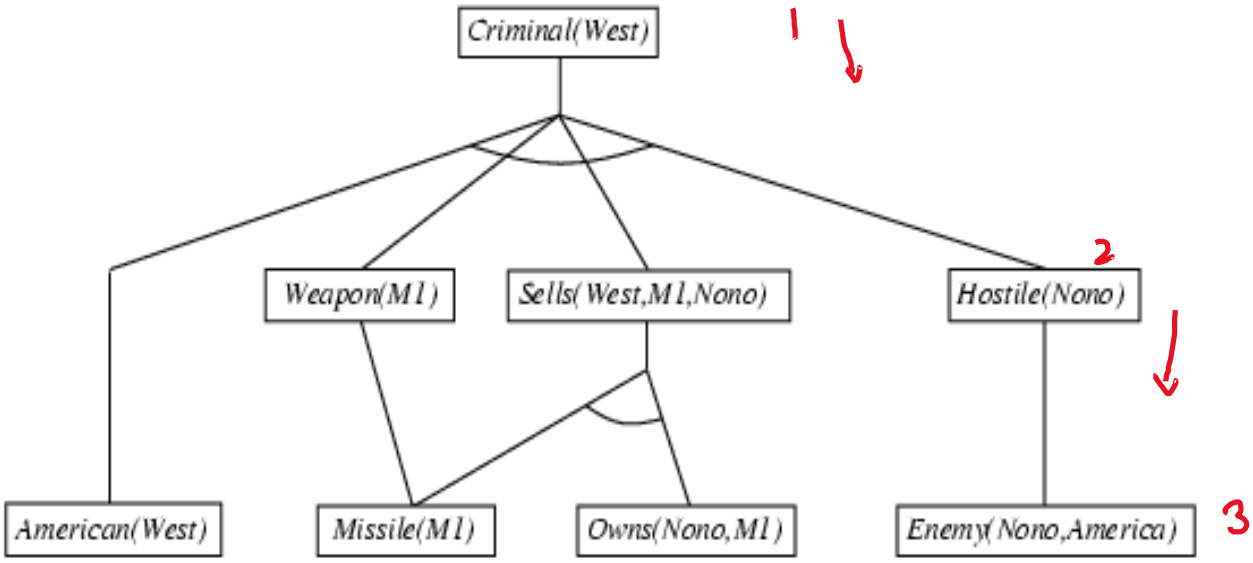
- Backward-chaining is based on modus ponens inference rule.

- In backward chaining, the goal is broken into sub-goal or sub-goals to prove the facts true.

- It is called a goal-driven approach, as a list of goals decides which rules are selected and used.

- Backward-chainingalgorithmisusedingametheory,automatedtheoremprovingtools,inference engines, proof assistants, and various AI applications.

- The backward-chaining method mostly used a **depth-first search** strategy for proof.



Proof tree constructed by backward chaining to prove that West is a criminal. The tree should be read depth first, left to right. To prove Criminal (West), we have to prove the four conjuncts below it. Some of these are in the knowledge base, and others require further backward chaining. Bindings for each successful unification are shown next to the corresponding sub goal. Note that once one sub goal in a conjunction succeeds, its substitution is applied to subsequent sub goals.

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.

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.

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.

we can infer facts Missile(M1), 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.

Then 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.

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 ∧ q → k.

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 ∧ q → k.

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,

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

Resolution

- **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

- 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

Tautologies
$(A \rightarrow B) \leftrightarrow (\sim A \vee B)$
$(A \vee (B \wedge C)) \leftrightarrow (A \vee B) \wedge (A \vee C)$

Example

- KB: $[P \rightarrow Q , Q \rightarrow R \wedge S]$
- KB in CNF: $[\sim P \vee Q , \sim Q \vee R , \sim Q \vee S]$
- Resolve KB(1) and KB(2) producing: $\sim P \vee R$ *(i.e., $P \rightarrow R$)*
- Resolve KB(1) and KB(3) producing: $\sim P \vee S$ *(i.e., $P \rightarrow S$)*
- New KB: $[\sim P \vee Q , \sim Q \vee \sim R \vee \sim S , \sim P \vee R , \sim P \vee S]$

- 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
- if P_j and $\neg Q_k$ **unify** with substitution list θ, then derive the resolvent sentence:

$subst(\theta, P_1 \vee \ldots \vee P_{j-1} \vee P_{j+1} \ldots P_n \vee Q_1 \vee \ldots Q_{k-1} \vee Q_{k+1} \vee \ldots \vee Q_m)$

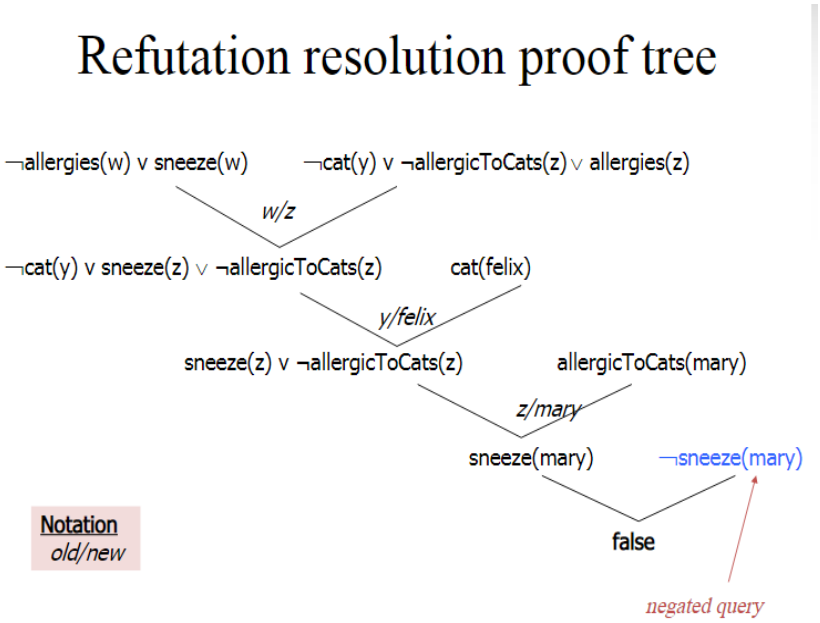
- Example
  - from clause **P(x, f(a)) ∨ P(x, f(y)) ∨ Q(y)**
  - and clause **¬P(z, f(a)) ∨ ¬Q(z)**
  - derive resolvent **P(z, f(y)) ∨ Q(y) ∨ ¬Q(z**
  - Using **θ = {x/z}**

Resolution refutation

- 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)

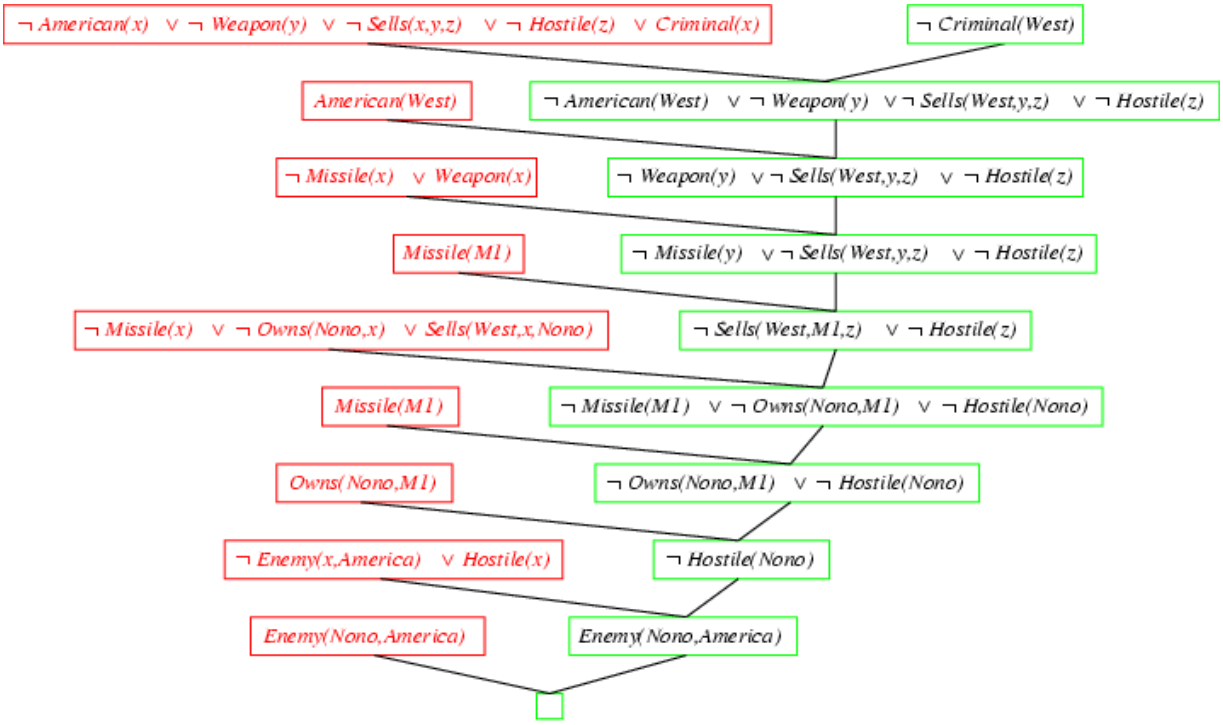


e crime example from Section 9.3. The sentences in CNF are

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

.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



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.

Natural Deduction Using Rules

Testing whether a proposition is a tautology by testing every possible truth assignment is expensive—there are exponentially many. We need a deductive system, which will allow us to construct proofs of tautologies in a step-by-step fashion.

The system we will use is known as natural deduction. The system consists of a set of rules of inference for deriving consequences from premises. One builds a proof tree whose root is the proposition to be proved and whose leaves are the initial assumptions or axioms (for proof trees, we usually draw the root at the bottom and the leaves at the top).

For example, one rule of our system is known as modus ponens. Intuitively, this says that if we know P is true, and we know that P implies Q, then we can conclude Q.

$$\frac{P \quad P \Rightarrow Q}{Q}$$

(modus ponens)

The propositions above the line are called premises; the proposition below the line is the conclusion. Both the premises and the conclusion may contain metavariables (in this case, P and Q) representing arbitrary propositions. When an inference rule is used as part of a proof, the metavariables are replaced in a consistent way with the appropriate kind of object (in this case, propositions).
Most rules come in one of two flavors: introduction or elimination rules. Introduction rules introduce the use of a logical operator, and elimination rules eliminate it. Modus ponens is an elimination rule for ⇒. On the right-hand side of a rule, we often write the name of the rule. This is helpful when reading proofs. In this case, we have written (modus ponens). We could also have written (⇒-elim) to indicate that this is the elimination rule for ⇒

Rules for Conjunction
Conjunction (∧) has an introduction rule and two elimination rules:

$$\frac{P \quad Q}{P \wedge Q}\ (\wedge\text{-intro}) \qquad \frac{P \wedge Q}{P}\ (\wedge\text{-elim-left}) \qquad \frac{P \wedge Q}{Q}\ (\wedge\text{-elim-right})$$

Rule for T
The simplest introduction rule is the one for T. It is called "unit". Because it has no premises, this rule is an **axiom**: something that can start a proof.

$$\frac{}{T}\ (\text{unit})$$

Rules for Implication
In natural deduction, to prove an implication of the form P ⇒ Q, we assume P, then reason under that assumption to try to derive Q. If we are successful, then we can conclude that P ⇒ Q.
In a proof, we are always allowed to introduce a new assumption P, then reason under that assumption. We must give the assumption a name; we have used the name x in the example below. Each distinct assumption must have a different name.

$$\frac{}{[x : P]}\ (\text{assum})$$

Because it has no premises, this rule can also start a proof. It can be used as if the proposition P were proved. The name of the assumption is also indicated here.
However, you do not get to make assumptions for free! To get a complete proof, all assumptions must be eventually *discharged*. This is done in the implication introduction rule. This rule introduces an implication $P \Rightarrow Q$ by discharging a prior assumption $[x : P]$. Intuitively, if Q can be proved under the assumption P, then the implication $P \Rightarrow Q$ holds without any assumptions. We write x in the rule name to show which assumption is discharged. This rule and modus ponens are the introduction and elimination rules for implications.

$$\frac{\begin{array}{c}[x : P] \\ \vdots \\ Q\end{array}}{P \Rightarrow Q}\ (\Rightarrow\text{-intro/x}) \qquad \frac{P \qquad P \Rightarrow Q}{Q}\ (\Rightarrow\text{-elim, modus ponens})$$

A proof is valid only if every assumption is eventually discharged. This must happen in the proof tree below the assumption. The same assumption can be used more than once.

Rules for Disjunction

$$\frac{P}{P \vee Q}\ (\vee\text{-intro-left}) \qquad \frac{Q}{P \vee Q}\ (\vee\text{-intro-right}) \qquad \frac{P \vee Q \qquad P \Rightarrow R \qquad Q \Rightarrow R}{R}\ (\vee\text{-elim})$$

Rules for Negation

A negation $\neg P$ can be considered an abbreviation for $P \Rightarrow \bot$:

$$\frac{P \Rightarrow \bot}{\neg P}\ (\neg\text{-intro}) \qquad \frac{\neg P}{P \Rightarrow \bot}\ (\neg\text{-elim})$$

A proposition that has a complete proof in a deductive system is called a **theorem** of that system.

Soundness and Completeness

A measure of a deductive system's power is whether it is powerful enough to prove all true statements. A deductive system is said to be **complete** if all true statements are theorems (have proofs in the system). For propositional logic and natural deduction, this means that all tautologies must have natural deduction proofs. Conversely, a deductive system is called **sound** if all theorems are true. The proof rules we have given above are in fact sound and complete for propositional logic: every theorem is a tautology, and every tautology is a theorem. Finding a proof for a given tautology can be difficult. But once the proof is found, checking that it is indeed a proof is completely mechanical, requiring no intelligence or insight whatsoever. It is therefore a very strong argument that the thing proved is in fact true.

We can also make writing proofs less tedious by adding more rules that provide reasoning shortcuts. These rules are sound if there is a way to convert a proof using them into a proof using the original rules. Such added rules are called **admissible**.

Procedural versus Declarative Knowledge

We have discussed various search techniques in previous units. Now we would consider a set of rules that represent,

1. Knowledge about relationships in the world and
2. 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.

In Procedural representation, knowledge is signified as procedures.

Example: Default reasoning and probabilistic reasoning are examples of procedural techniques.

Here, heuristic knowledge of "How to do things efficiently "can be simply signified.

- Control information essential to use the knowledge is entrenched in the knowledge itself. For example, how to discover relevant facts, make inferences etc.
- Needs an interpreter to follow instructions provided in knowledge.

  Example: Let us assume what knowledge an alphabetical sorter would want:

- Implicit knowledge that A occurs before B etc.
- This is simple really integer comparison of (ASCII) codes for A, B.

- All programs enclose procedural knowledge of this sort.
- The procedural information here is that knowledge of how to alphabetise is signified explicitly in the alphabetization procedure.

- A declarative system might have to have explicit details such as A occurs before B, B comes before C etc.

1. 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.
2. Logic Programming – Representing Knowledge Using Rules
   - Logic programming is a programming paradigm in which logical assertions viewed as programs.
   - These are several logic programming systems, PROLOG is one of them.
   - A PROLOG program consists of several logical assertions where each is a horn clause

     i.e. a clause with at most one positive literal.

- ▪
  - • Ex : P, P V Q, P → Q
  - • The facts are represented on Horn Clause for two reasons.
    0. Because of a uniform representation, a simple and efficient interpreter can write.
    1. The logic of Horn Clause decidable.

- Also, The first two differences are the fact that PROLOG programs are actually sets of Horn clause that have been transformed as follows:-
  0. If the Horn Clause contains no negative literal then leave it as it is.
  1. Also, Otherwise rewrite the Horn clauses as an implication, combining all of the

     negative literals into the antecedent of the implications and the single positive

     literal into the consequent.

- • Moreover, This procedure causes a clause which originally consisted of a disjunction of

  literals (one of them was positive) to be transformed into a single implication whose

  antecedent is a conjunction universally quantified.

- • But when we apply this transformation, any variables that occurred in negative literals

  and so now occur in the antecedent become existentially quantified, while the variables in

  the consequent are still universally quantified.
  For example the PROLOG clause P(x): – Q(x, y) is equal to logical expression ∀x: ∃y: Q (x, y) → P(x).

- • The difference between the logic and PROLOG representation is that the PROLOG interpretation has a fixed control strategy. And so, the assertions in the PROLOG program define a particular search path to answer any question.
- • But, the logical assertions define only the set of answers but not about how to choose among those answers if there is more than one.

  Consider the following example: 1. Logical representation

  ∀x : pet(x) ∧ small (x) → apartmentpet(x)

  ∀x : cat(x) ∧ dog(x) → pet(x)
  ∀x : poodle (x) → dog (x) ∧ small (x)

  poodle (fluffy)

  2. Prolog representation

  apartmentpet (x) : pet(x), small (x) pet (x): cat (x)
  pet (x): dog(x)

dog(x): poodle (x) small (x): poodle(x)

poodle (fluffy)

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)**
-

- Indexing
  - ❖ A large number of rules ⇒ too sluggish to locate a rule
  - ❖ Indexing: Use the existing state as an index into rules and choose the matching ones

    instantly
  - ❖ Only functions when preconditions of rules match accurately
  - ❖ Only functions when preconditions of rules match exact board configuration
  - ❖ It's not forever apparent whether a rule's preconditions are pleased by a specific state

- ❖ There's a swapping among the ease of writing rules (high-level descriptions) and the ease of the matching process
- Matching with variables

❖ Generality in the statements of the rules: Require a search process to determine a match among a particular state and the preconditions of a specified rule.

- Backward-chaining systems
  - ❖ One-one matching algorithm
    - ♦ Unification procedure + Depth-first backtracking to select individual rules
    - ♦ Forward-chaining systems
  - ❖ Many-many matching algorithm: RETE
- Approximate matching

❖ Rules should be applied if their preconditions roughly match the present situation

Example: A speech-understanding program

- Rules: An explanation of a physical waveform to phones (a, e, ...)
- Physical signal: differences in the way individuals speak, result of background noise, etc.
- Conflict resolution:
  - ❖ Preferences dependent on rules:
    - ♦ Specificity of rules
    - ♦ Physical order of rules
  - ❖ Preferences dependent on objects:
    - ♦ Importance of objects
    - ♦ Position of objects
  - ❖ Preferences dependent on states:

♦ Assessment of states

6.4.2 Control Knowledge

An algorithm comprises: logic component, that mentions the knowledge to be utilized in solving problems, and manage component, that identifies the problem-solving approaches by means of which that knowledge is utilized.

Therefore, Algorithm = Logic + Control. The logic component identifies the meaning of the algorithm while the control component only affects its competence. An algorithm may be formulated in dissimilar manners, generating similar behavior. One formulation, may have a apparent statement in logic component but utilize a complicated problem solving strategy in the

control component. The other formulation may have a complex logic component but utilize a simple problem-solving approach. The competence of an algorithm can frequently be improved by enhancing the control component without altering the logic of the algorithm and thus without altering the meaning of the algorithm.

The approach in databases is towards the division of logic and control. The programming languages these days do not differentiate between them.

Computer programs will be more frequently accurate, more simply enhanced, and more readily adapted to new troubles when programming languages divide logic and control, and when execution mechanisms offer more powerful problem-solving amenities of the type given by intelligent theorem-proving systems.