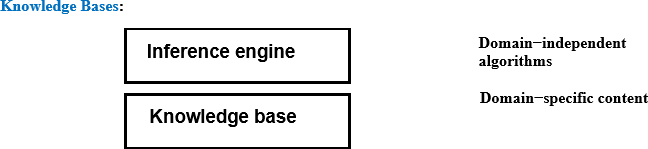
# UNIT II

# CHAPTER 3&4

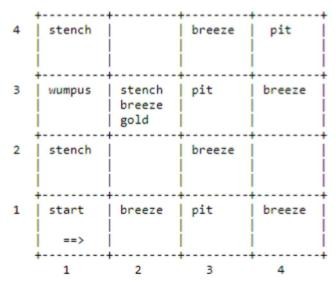
Knowledge Based Agents A knowledge-based agent needs a KB and an inference mechanism. It operates by storing sentences in its knowledge base, inferring new sentences with the inference mechanism, and using them to deduce which actions to take The

interpretation of a sentence is the fact to which it refers.



Knowledge base = set of sentences in a formal language Declarative approach to building an agent (or other system): Tell it what it needs toknow - Thenitcan Askitselfwhattodo— answersshouldfollowfromtheKB Agents can be viewed at the knowledge leveli.e., what they know, regardless of howimplemented or at the implementation leveli.e.,data structuresinKBand algorithmsthatmanipulatethem. The Wumpus World:

A variety of "worlds" are being used as examples for Knowledge Representation, Reasoning, and Planning. Among them the Vacuum World, the Block World, and the Wumpus World. The Wumpus World was introduced by Genesereth, and is discussed in Russell-Norvig. The Wumpus World is a simple world (as is the Block World) for which to represent knowledge and to reason. It is a cave with a number of rooms, represented as a 4x4 square



Rules of the Wumpus World The neighborhood of a node consists of the four squares north, south, east, and west of the given square. In a square the agent gets a vector of percepts, with components Stench, Breeze, Glitter, Bump, Scream For example [Stench, None, Glitter, None, None]  Stench is perceived at a square iff the wumpus is at this square or in its neighborhood.  Breeze is perceived at a square iff a pit is in the neighborhood of this square.  Glitter is perceived at a square iff gold is in this square  Bump is perceived at a square iff the agent goes Forward into a wall  Scream is perceived at a square iff the wumpus is killed anywhere in the cave An agent can do the following actions (one at a time): Turn (Right), Turn (Left), Forward, Shoot, Grab, Release, Climb  The agent can go forward in the direction it is currently facing, or Turn Right, or Turn Left. Going forward into a wall will generate a Bump percept.  The agent has a single arrow that it can shoot. It will go straight in the direction faced by the agent until it hits (and kills) the wumpus, or hits (and is absorbed by) a wall.  The agent can grab a portable object at the current square or it can Release an object that it is holding.  The agent can climb out of the cave if at the Start square.The Start square is (1,1) and initially the agent is facing east. The agent dies if it is in the same square asthe wumpus. The objective of the game is to kill the wumpus, to pick up the gold, and to climb out with it. Representing our Knowledge about the Wumpus World Percept(x, y) Where x must be a percept vector and y must be a situation. It means that at situation y theagentperceives x.For convenience we introduce the following definitions: 

Percept([Stench,y,z,w,v],t) = > Stench(t)  Percept([x,Breeze,z,w,v],t) = > Breeze(t)  Percept([x,y,Glitter,w,v],t) = > AtGold(t) Holding(x, y)

Where x is an object and y is a situation. It means that the agent is holding the object x in situation y. Action(x, y) Where x must be an action (i.e. Turn (Right), Turn (Left), Forward,) and y must be a situation. It means that at situation y the agent takes action x. At(x,y,z) Where x is an object, y is a Location, i.e. a pair [u,v] with u and v in {1, 2, 3, 4}, and z is a situation. It means that the agent x in situation z is at location y. Present(x,s) Means that object x is in the current room in the situation s. Result(x, y) It means that the result of applying action x to the situation y is the situation Result(x,y).Notethat Result(x,y) is a term, not a statement. For example we can say  Result(Forward, S0) = S1  Result(Turn(Right),S1) = S2 These definitions could be made more general. Since in the Wumpus World there is a single agent, there is no reason for us to make predicates and functions relative to a specific agent. In other"worlds" we should change things appropriately.

Validity And Satisfiability A sentence is valid

if it is true in all models, e.g.,True,A∨¬A, A⇒A,(A𝖠(A⇒B)) ⇒B Validity is connected to inference via the Deduction Theorem: KB |= αif and onlyif(KB⇒α) isvalid Asentenceissatisfiableifitistrue insome model e.g., A∨B, C Asentence isunsatisfiableifitistrueinnomodels e.g., A 𝖠¬A Satisfiability is connected to inference via the following: KB|=α iff(KB𝖠¬α)isunsatisfiable i.e., prove α by reductionandabsurdum

Proof Methods

Proof methods divide into (roughly)two kinds:

Application of inference rules – Legitimate(sound)generationofnewsentencesfromold – Proof=asequenceofinferenceruleapplicationscanuseinferencerulesasoperatorsinastand

|  |  |  |  |
| --- | --- | --- | --- |
| ardsearch | algorithm | – Typicallyrequiretranslationofsentencesintoanormalform | Model |
| checking | – | Truthtableenumeration(alwaysexponentialinn) | – |

Improvedbacktracking,e.g.,Davis–Putnam–Loge Mann–Loveland – Heuristic searchinmodelspace(soundbutincomplete) e.g.,min-conflicts-likehillclimbingalgorithms

Forward and Backward Chaining

Horn Form (restricted) KB = conjunction of Horn clauses Horn clause = – proposition symbol;or – (conjunctionofsymbols) ⇒ symbol Example KB: C𝖠(B ⇒ A) 𝖠 (C𝖠D ⇒ B)

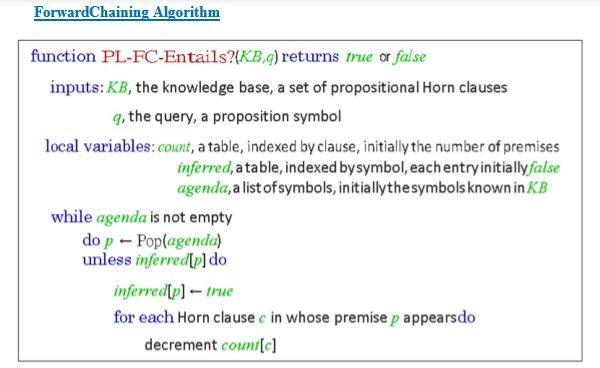
Modus Ponens (for Horn Form): complete for Horn KBs α1,...,αn,α1𝖠···𝖠α⇒ β β

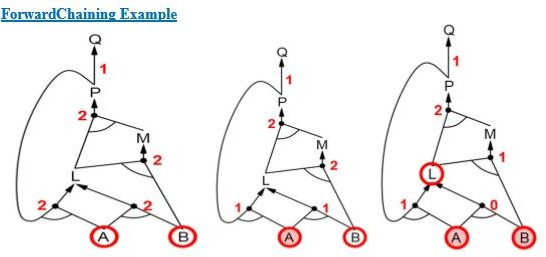
Canbeusedwithforwardchaining orbackwardchaining. These algorithms areverynaturalandruninlineartime.,

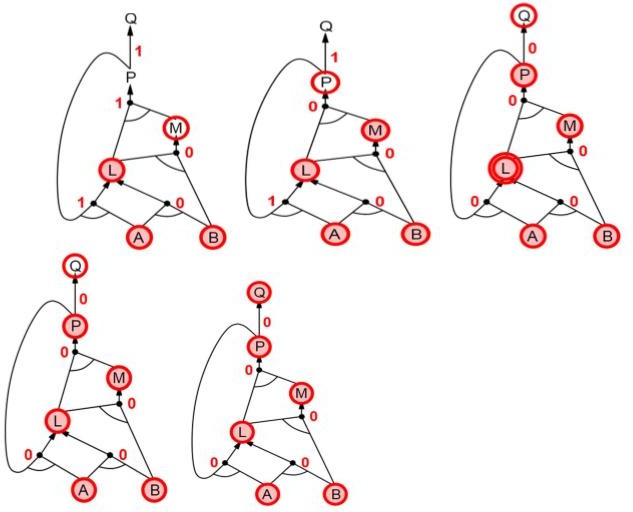
ForwardChaining

Idea: If anyrulewhosepremisesaresatisfiedintheKB, additsconclusiontotheKB,untilqueryisfound

ForwardChaining Algorithm







Proof of Completeness

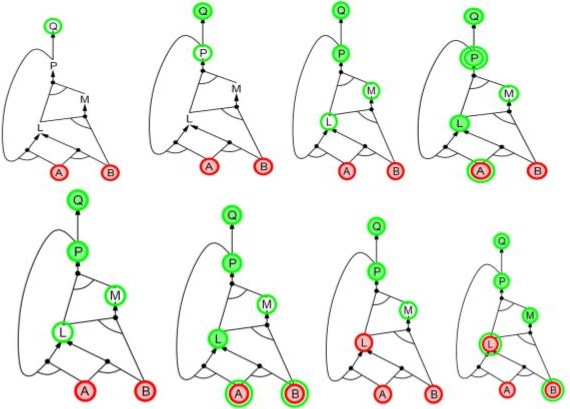
FC derives every atomic sentence that is entailed by KB

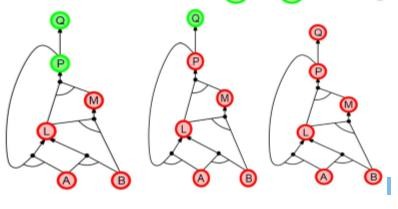
* 1. FCreachesafixedpointwherenonewatomicsentencesarederived
  2. Considerthefinalstateasamodelm,assigningtrue/falsetosymbols
  3. Every clause in the original KB is true inm i. Proof:Supposeaclausea1𝖠...𝖠ak⇒bisfalsei

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| nm | Then a1𝖠. | . . 𝖠akis | true | in | m | and | b | is | false in m |
| Thereforethealgorithmhasnotreachedafixedpoint ! | | | | | 4. Hence | | m is a | | model ofKB 5. |
| IfKB|=q,thenqistrueineverymodelofKB,includingm | | | | |  | | a. | | Generalidea: |
| constructanymodelofKBby soundinference,checkα | | | | |  | |  | |  |

Backward Chaining

Idea:workbackwardsfromthequeryq: to prove q byBC, check if q is known already, or prove by BC all premises of some rule concluding q Avoidloops: checkifnewsubgoalisalreadyonthegoalstack Avoid repeated work: check if new subgoal 1. has already been proved true, or 2. has alreadyfailed





Forward vs Backward Chaining

FC is data-driven, cf. automatic, unconscious processing, e.g.,objectrecognition,routinedecisions Maydolotsofworkthatisirrelevanttothegoal BC is goal- driven, appropriate forproblem-solving, e.g., Where are my keys? How do I get into a PhD program? Complexity of BC can be much less than linear in size of KB

**FIRST ORDER LOGIC:**

**PROCEDURAL LANGUAGES AND PROPOSITIONAL LOGIC:**

Drawbacks of Procedural Languages

 Programming languages (such as C++ or Java or Lisp) are by far the largest class of formal languages in common use. Programs themselves represent only computational processes. Data structures within programs can represent facts.

For example, a program could use a 4 × 4 array to represent the contents of the wumpus world. Thus, the programming language statement World\*2,2+← Pit is a fairly natural way to assert that there is a pit in square [2,2].

What programming languages lack is any general mechanism for deriving facts from other facts; each update to a data structure is done by a domain-specific procedure whose details are derived by the programmer from his or her own knowledge of the domain.

 A second drawback of is the lack the expressiveness required to handle partial information . For example data structures in programs lack the easy way to say, “There is a pit in \*2,2+ or \*3,1+” or “If the wumpus is in \*1,1+ then he is not in \*2,2+.”

Advantages of Propositional Logic

 The declarative nature of propositional logic, specify that knowledge and inference are separate, and inference is entirely domain-independent.  Propositional logic is a declarative language because its semantics is based on a truth relation between sentences and possible worlds.  It also has sufficient expressive power to deal with partial information, using disjunction and negation.

 Propositional logic has a third COMPOSITIONALITY property that is desirable in representation languages, namely, compositionality. In a compositional language, the meaning of a sentence is a function of the meaning of its parts. For example, the meaning of “S1,4𝖠 S1,2” is related to the meanings of “S1,4” and “S1,2.

Drawbacks of Propositional Logic  Propositional logic lacks the expressive power to concisely describe an environment with many objects.

For example, we were forced to write a separate rule about breezes and pits for each square, such as B1,1⇔ (P1,2 ∨ P2,1) .

 In English, it seems easy enough to say, “Squares adjacent to pits are breezy.”  The syntax and semantics of English somehow make it possible to describe the environment concisely

SYNTAX AND SEMANTICS OF FIRST-ORDER LOGIC

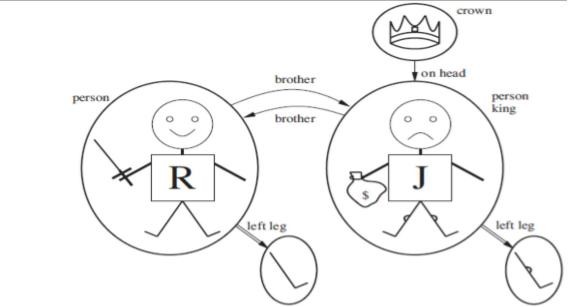
Models for first-order logic :

The models of a logical language are the formal structures that constitute the possible worlds under consideration. Each model links the vocabulary of the logical sentences to elements of the possible world, so that the truth of any sentence can be determined. Thus, models for propositional logic link proposition symbols to predefined truth values. Models for first-order logic have objects. The domain of a model is the set of objects or domain elements it contains. The domain is required to be nonempty—every possible world must contain at least one object.

A relation is just the set of tuples of objects that are related.  Unary Relation: Relations relates to single Object  Binary Relation: Relation Relates to multiple objects Certain kinds of relationships are best considered as functions, in that a given object must be related to exactly one object.

For Example:

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



Unary Relation : John is a king Binary Relation :crown is on head of john , Richard is brother ofjohn The unary "left leg" function includes the following mappings: (Richard the Lionheart) ->Richard's left leg (King John) ->Johns left Leg

Symbols and interpretations

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 betweenthe 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 possiblemodels 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. Hence, the number of possible models is anbounded, as is the number of interpretations.

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.

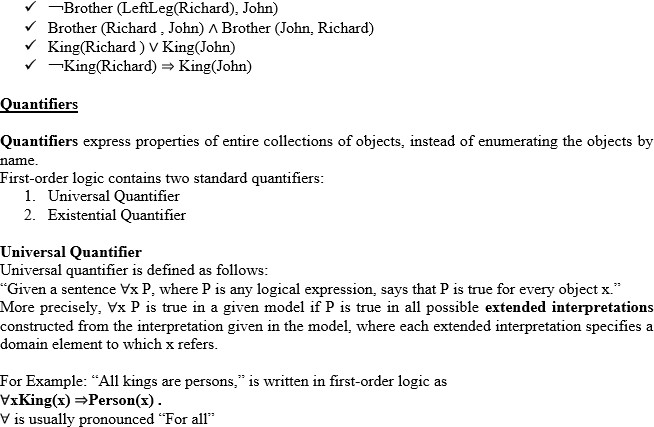
Atomic sentences

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:



Thus, the sentence says, ―For all x, if x is a king, then x is a person.‖ The symbol x is called a variable. Variables are lowercase letters. A variable is a term all by itself, and can also serve as the argument of a function A term with no variables is called a ground term.

Assume we can extend the interpretation in different ways: x→ Richard the Lionheart, x→ King John, x→ Richard’s left leg, x→ John’s left leg, x→ the crown

The universally quantified sentence ∀x King(x) ⇒Person(x) is true in the original model if the sentence King(x) ⇒Person(x) is true under each of the five extended interpretations. That is, the universally quantified sentence is equivalent to asserting the following five sentences:

Richard the Lionheart 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.

##### 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.

“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 if P is true in at least one extended interpretationthat assigns x to a domain element.” ∃x is pronounced “There exists an x such that . . .” or “For some x . . .”.

For example, that King John has a crown on his head, we write ∃xCrown(x) 𝖠OnHead(x, John) Given assertions:

Richard the Lionheart is a crown 𝖠Richard the Lionheart is on John’s head; King John is a crown

𝖠King John is on John’s head; Richard’s left leg is a crown 𝖠Richard’s left leg is on John’s head; John’s left leg is a crown 𝖠John’s left leg is on John’s head; The crown is a crown 𝖠the 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. Just as ⇒appears to be the natural connective to use with ∀, 𝖠is the natural connective to use with ∃.

Nested quantifiers

One can express more complex sentences using multiple quantifiers.

For example, “Brothers are siblings” can be written as ∀x∀y Brother (x, y) ⇒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∀x, y Sibling(x, y) ⇔Sibling(y, x).

In other cases we will have mixtures.

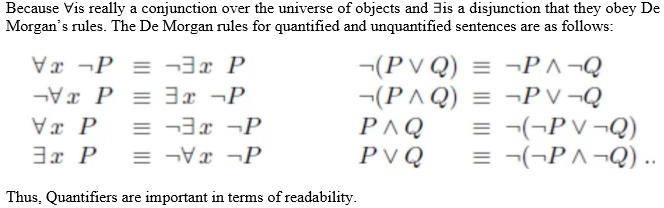
For example: 1. “Everybody loves somebody” means that for every person, there is someone that person loves: ∀x∃y Loves(x, y) . 2. On the other hand, to say “There is someone who is loved by everyone,” we write ∃y∀x Loves(x, y) .

Connections between ∀and ∃

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)”. 2. “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:



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.

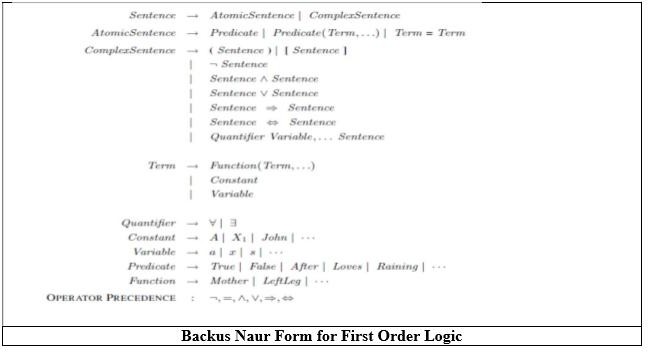
For example,

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

) 𝖠￢(x=y) .

The sentence

∃x, y Brother (x,Richard ) 𝖠Brother (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.



USING FIRST ORDER LOGIC Assertions and queries in first-order logic Assertions:

Sentences are added to a knowledge base using TELL, exactly as in propositional logic. Such sentences are called assertions.

For example,

John is a king, TELL (KB, King (John)). Richard is a person. TELL (KB, Person (Richard)). All kings are persons: TELL (KB, ∀x King(x) ⇒Person(x)).

Asking Queries:

We can ask questions of the knowledge base using ASK. Questions asked with ASK are called queries or goals.

For example,

ASK (KB, King (John)) returns true.

Anyquery that is logically entailed by the knowledge base should be answered affirmatively. Forexample, given the two preceding assertions, the query:

“ASK (KB, Person (John))” should also return true. Substitution or binding list

We can ask quantified queries, such as ASK (KB, ∃x Person(x)) .

The answer is true, but this is perhaps not as helpful as we would like. It is rather like answering “Can you tell me the time?” with “Yes.”

If we want to know what value of x makes the sentence true, we will need a different function, ASKVARS, which we call with ASKVARS (KB, Person(x)) and which yields a stream of answers.

In this case there will be two answers: {x/John} and {x/Richard}. Such an answer is called a substitution or binding list.

ASKVARS is usually reserved for knowledge bases consisting solely of Horn clauses, because in such knowledge bases every way of making the query true will bind the variables to specific values.

The kinship domain

The objects in Kinship domain are people.

We have two unary predicates, Male and Female.

Kinship relations—parenthood, brotherhood, marriage, and so on—are represented by binary predicates: Parent, Sibling, Brother,Sister,Child, Daughter, Son, Spouse, Wife, Husband, Grandparent,Grandchild, Cousin, Aunt, and Uncle.

We use functions for Mother and Father, because every person has exactly one of each of these.

We can represent each function and predicate, writing down what we know in termsof the other symbols.

For example:- 1. one’s mother is one’s female parent: ∀m, c Mother (c)=m ⇔Female(m) 𝖠Parent(m, c) .

1. One’s husband is one’s male spouse: ∀w, h Husband(h,w) ⇔Male(h) 𝖠Spouse(h,w) .
2. Male and female are disjoint categories: ∀xMale(x) ⇔￢Female(x) .
3. Parent and child are inverse relations: ∀p, c Parent(p, c) ⇔Child (c, p) .
4. A grandparent is a parent of one’s parent: ∀g, c Grandparent (g, c) ⇔∃p Parent(g, p) 𝖠Parent(p, c)

.

1. A sibling is another child of one’s parents: ∀x, y Sibling(x, y) ⇔x \_= y 𝖠∃p Parent(p, x) 𝖠Parent(p, y) .

Axioms:

Each of these sentences can be viewed as an axiom of the kinship domain. Axioms are commonly associated with purely mathematical domains. They provide the basic factual information from which useful conclusions can be derived.

Kinship axioms are also definitions; they have the form ∀x, y P(x, y) ⇔. . ..

The axioms define the Mother function, Husband, Male, Parent, Grandparent, and Sibling predicates in terms of other predicates.

Our definitions “bottom out” at a basic set of predicates (Child, Spouse, and Female) in terms of which the others are ultimately defined. This is a natural way in which to build up the representation of a domain, and it is analogous to the way in which software packages are built up by successive definitions of subroutines from primitive library functions.

Theorems:

Not all logical sentences about a domain are axioms. Some are theorems—that is, they are entailed by the axioms.

For example, consider the assertion that siblinghood is symmetric: ∀x, y Sibling(x, y) ⇔Sibling(y, x) .

It is a theorem that follows logically from the axiom that defines siblinghood. If we ASK the knowledge base this sentence, it should return true. From a purely logical point of view, a knowledge base need contain only axioms and no theorems, because the theorems do not increase the set of conclusions that follow from the knowledge base. From a practical point of view, theorems are essential to reduce the computational cost of deriving new sentences. Without them, a reasoning system has to start from first principles every time.

Axioms :Axioms without Definition

Not all axioms are definitions. Some provide more general information about certain predicates without constituting a definition. Indeed, some predicates have no complete definition because we do not know enough to characterize them fully.

For example, there is no obvious definitive way to complete the sentence

∀xPerson(x) ⇔. . .

Fortunately, first-order logic allows us to make use of the Person predicate without completely defining it. Instead, we can write partial specifications of properties that every person has and properties that make something a person:

∀xPerson(x) ⇒. . . ∀x . . . ⇒Person(x) .

Axioms can also be “just plain facts,” such as Male (Jim) and Spouse (Jim, Laura).Such facts form the descriptions of specific problem instances, enabling specific questions to be answered. The answers to these questions will then be theorems that follow from the axioms

Numbers, sets, and lists Number theory

Numbers are perhaps the most vivid example of how a large theory can be built up from NATURAL NUMBERS a tiny kernel of axioms. We describe here the theory of natural numbers or non-negative integers. We need:

 predicate NatNum that will be true of natural numbers;  one PEANO AXIOMS constant symbol, 0;

 One function symbol, S (successor).  The Peano axioms define natural numbers and addition.

Natural numbers are defined recursively: NatNum(0) . ∀n NatNum(n) ⇒ NatNum(S(n)) .

That is, 0 is a natural number, and for every object n, if n is a natural number, then S(n) is a natural number.

So the natural numbers are 0, S(0), S(S(0)), and so on. We also need axioms to constrain the successor function: ∀n 0 != S(n) . ∀m, n m != n ⇒ S(m) != S(n) .

Now we can define addition in terms of the successor function: ∀m NatNum(m) ⇒ + (0, m) = m .

∀m, n NatNum(m) 𝖠 NatNum(n) ⇒ + (S(m), n) = S(+(m, n))

The first of these axioms says that adding 0 to any natural number m gives m itself. Addition is represented using the binary function symbol “+” in the term + (m, 0);

To make our sentences about numbers easier to read, we allow the use of infix notation. We can also write S(n) as n + 1, so the second axiom becomes :

∀m, n NatNum (m) 𝖠 NatNum(n) ⇒ (m + 1) + n = (m + n)+1 .

This axiom reduces addition to repeated application of the successor function. Once we have addition, it is straightforward to define multiplication as repeated addition, exponentiation as repeated multiplication, integer division and remainders, prime numbers, and so on. Thus, the whole of number theory (including cryptography) can be built up from one constant, one function, one predicate and four axioms.

Sets

The domain of sets is also fundamental to mathematics as well as to commonsense reasoning. Sets can be represented as individualsets, including empty sets.

Sets can be built up by:  adding an element to a set or  Taking the union or intersection of two sets.

Operations that can be performed on sets are:  To know whether an element is a member of a set

 Distinguish sets from objects that are not sets. Vocabulary of set theory:

The empty set is a constant written as { }. There is one unary predicate, Set, which is true of sets. The binary predicates are

 ∈x s (x is a member of set s)  s1⊆ s2 (set s1 is a subset, not necessarily proper, of set s2).

The binary functions are

 s1 ∩ s2 (the intersection of two sets),  s𝖴1 s2 (the union of two sets), and  ,x|s- (the set resulting from adjoining element x to set s).

One possible set of axioms is as follows:

 The only sets are the empty set and those made by adjoining something to a set:∀sSet(s) ⇔(s={})

∨(∃x, s2 Set(s2) 𝖠s={x|s2}) .  The empty set has no elements adjoined into it. In other words, there is no way to decompose {} into a smaller set and an element: ￢∃x, s {x|s}={} .  Adjoining an element already in the set has no effect: ∀x, s x∈s ⇔s={x|s} .  The only members of a set are the elements that were adjoined into it. We express this recursively, saying that x is a member of s if and only if s is equal to some set s2 adjoined with some element y, where either y is the same as x or x is a member of s2: ∀x, s x∈s ⇔∃y, s2 (s={y|s2} 𝖠(x=y ∨x∈s2))  A set is a subset of another set if and only if all of the first set’s members are members of the second set: ∀s1, s2 s1 ⊆s2 ⇔(∀x x∈s1

⇒x∈s2)  Two sets are equal if and only if each is a subset of the other: ∀s1, s2 (s1 =s2) ⇔(s1 ⊆s2

𝖠s2 ⊆s1)

 An object is in the intersection of two sets if and only if it is a member of both sets:∀x, s1, s2 x∈(s1 ∩ s2) ⇔(x∈s1 𝖠x∈s2)  An object is in the union of two sets if and only if it is a member of either set: ∀x, s1, s2 x∈(s1 𝖴s2) ⇔(x∈s1 ∨x∈s2)

Lists : are similar to sets. The differences are that lists are ordered and the same element canappear more than once in a list. We can use the vocabulary of Lisp for lists:

 Nil is the constant list with no element;s  Cons, Append, First, and Rest are functions;  Find is the predicate that does for lists what Member does for sets.  List? is a predicate that is true only of lists.  The empty list is \* +.  The term Cons(x, y), where y is a nonempty list, is wittren [x|y].  The

term Cons(x, Nil) (i.e., the list containing the element x) is written as [x].  A list of several elements, such as [A,B,C], corresponds to the nested term  Cons(A, Cons(B, Cons(C, Nil))).

The wumpus world

Agents Percepts and Actions

The wumpus agent receives a percept vector with five elements. The corresponding first-order sentence stored in the knowledge base must include both the percept and the time at which it occurred; otherwise, the agent will get confused about when it saw what.We use integers for time steps. A typical percept sentence would be

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

Here, Percept is a binary predicate, and Stench and so on are constants placed in a list. The actions in the wumpus world can be represented by logical terms:

Turn (Right), Turn (Left), Forward,Shoot,Grab, Climb.

To determine which is best, the agent program executes the query:

ASKVARS (∃a BestAction (a, 5)), which returns a binding list such as {a/Grab}. The agent program can then return Grab as the action to take.

The raw percept data implies certain facts about the current state.

For example: ∀t, s, g, m, c Percept ([s, Breeze, g,m, c], t) ⇒Breeze(t) , ∀t, s, b, m, c Percept ([s, b, Glitter,m, c], t) ⇒Glitter (t) ,

# UNIT III – Knowledge and Reasoning

These rules exhibit a trivial form of the reasoning process called perception.

Simple ―reflex‖ behavior can also be implemented by quantified implication sentences.

For example, we have ∀tGlitter (t) ⇒BestAction(Grab, t) .

Given the percept and rules from the preceding paragraphs, this would yield the desired conclusion Best Action (Grab, 5)—that is, Grab is the right thing to do.

Environment Representation

Objects are squares, pits, and the wumpus. Each square could be named—Square1,2and so on—but then the fact that Square1,2and Square1,3 are adjacent would have to be an ―extra‖ fact, and this needs one suchfact for each pair of squares. It is better to use a complex term in which the row and columnappear as integers;

For example, we can simply use the list term [1, 2]. Adjacency of any two squares can be defined as:

∀x, y, a, b Adjacent ([x, y], [a, b]) ⇔ (x = a 𝖠(y = b − 1 ∨y = b + 1)) ∨(y = b 𝖠(x = a − 1 ∨x

= a + 1)).

Each pit need not be distinguished with each other. The unary predicate Pit is true of squares containing pits.

Since there is exactly one wumpus, a constant Wumpus is just as good as a unary predicate. The agent’s location changes over time, so we write At (Agent, s, t) to mean that theagent is at square s at time t.

To specify the Wumpus location (for example) at [2, 2] we can write ∀t At (Wumpus, [2, 2], t).

Objects can only be at one location at a time: ∀x, s1, s2, t At(x, s1, t) 𝖠At(x, s2, t) ⇒s1 = s2 .

Given its current location, the agent can infer properties of the square from properties of its current percept.

For example, if the agent is at a square and perceives a breeze, then that square is breezy:

∀s, t At(Agent, s, t) 𝖠Breeze(t) ⇒Breezy(s) .

It is useful to know that a square is breezy because we know that the pits cannot move about. Breezy has no time argument.

Having discovered which places are breezy (or smelly) and, very importantly, not breezy (or not smelly), the agent can deduce where the pits =e (and where the wumpus is).

There are two kinds of synchronic rules that could allow such deductions: Diagnostic rules:

Diagnostic rules lead from observed effects to hidden causes. For finding pits, the obvious diagnostic rules say that if a square is breezy, some adjacent square must contain a pit, or

∀s Breezy(s) ⇒∃r Adjacent (r, s)𝖠Pit(r) ,

and that if a square is not breezy, no adjacent square contains a pit: ∀s￢Breezy (s) ⇒￢∃r Adjacent (r, s) 𝖠 Pit (,r) .Combining these two, we obtain the biconditional sentence ∀s Breezy ( s )⇔∃r Adjacent(r, s) 𝖠 Pit (r) .

Causal rules:

Causal rules reflect the assumed direction of causality in the world: some hidden property of the world causes certain percepts to be generated. For example, a pit causes all adjacent squares to be breezy:

and if all squares adjacent to a given square are pitless, the square will not be breezy: ∀s[∀r Adjacent (r, s) ⇒￢Pit (r)] ⇒￢Breezy ( s ) .

It is possible to show that these two sentences together are logically equivalent to the biconditional sentence ― ∀s Breezy ( s )⇔∃r Adjacent(r, s) 𝖠 Pit (r)‖ .

The biconditional itself can also be thought of as causal, because it states how the truth value of Breezy is generated from the world state.

Systems that reason with causal rules are called model-based reasoning systems, because the causal rules form a model of how the environment operates.

Whichever kind of representation the agent uses, ifthe axioms correctly and completely describe the way the world works and the way that percepts are produced, then any complete logical inference procedure will infer the strongest possible description of the world state, given the available percepts. Thus, the agent designer can concentrate on getting the knowledgeright, without worrying too much about the processes of deduction.

***Inference in First-Order Logic***

***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 θto the sentence *a.* Then the rule is written

For any variable v and ground term *g.*

For example, there is a sentence in knowledge base stating that all greedy kings are Evils



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



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,

For example, from the sentence



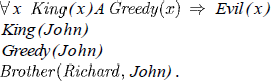
So, we can infer the sentence



As long as *C1* 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 oldin the sense that it is satisfiable exactly when the original knowledge base is satisfiable.

###### 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

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

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).*

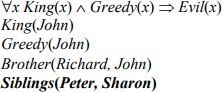
###### Disadvantage:

If the knowledge base includes a function symbol, the set of possible ground term substitutions is infinite. Propositional algorithms will have difficulty with an infinitely large set of sentences.

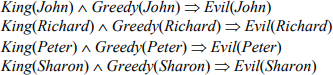
NOTE:

Entailment for first-order logic is *semi decidable* which means algorithms exist that say yes to every entailed sentence, but no algorithm exists that also says no to every non entailed sentence

* 1. ***Unification and Lifting***

Consider the above discussed example, if we add Siblings (Peter, Sharon) to the knowledge base then it will be

Removing Universal Quantifier will add new sentences to the knowledge base which are not necessary for the query *Evil (John)?*



Hence we need to teach the computer to make better inferences. For this purpose Inference rules were used.

###### 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 Generalized 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** 𝖠 **p2** 𝖠 **…** 𝖠 **pn** ⇒ **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.

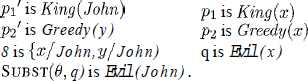
Suppose that instead of knowing Greedy (John) in our example we know that everyone is greedy:

##### ∀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,



***Unification:***

It is the process used to find substitutions that make different logical expressions look identical.

**Unification** is a key component of all first-order Inference algorithms.

UNIFY (p, q) = θ where SUBST (θ, p) = SUBST (θ, q) θ is our unifier value (if one exists). Ex: ―Who does John know?‖

UNIFY (Knows (John, x), Knows (John, Jane)) = {x/ Jane}. UNIFY (Knows (John, x), Knows (y, Bill)) = {x/Bill, y/ John}.

UNIFY (Knows (John, x), Knows (y, Mother(y))) = {x/Bill, y/ John} UNIFY (Knows (John, x), Knows (x, Elizabeth)) = FAIL

* The last unification fails because both use the same variable, X. X can’t equal both John and Elizabeth. To avoid this change the variable X to Y (or any other value) in Knows(X, Elizabeth)

##### Knows(X, Elizabeth) → Knows(Y, Elizabeth)

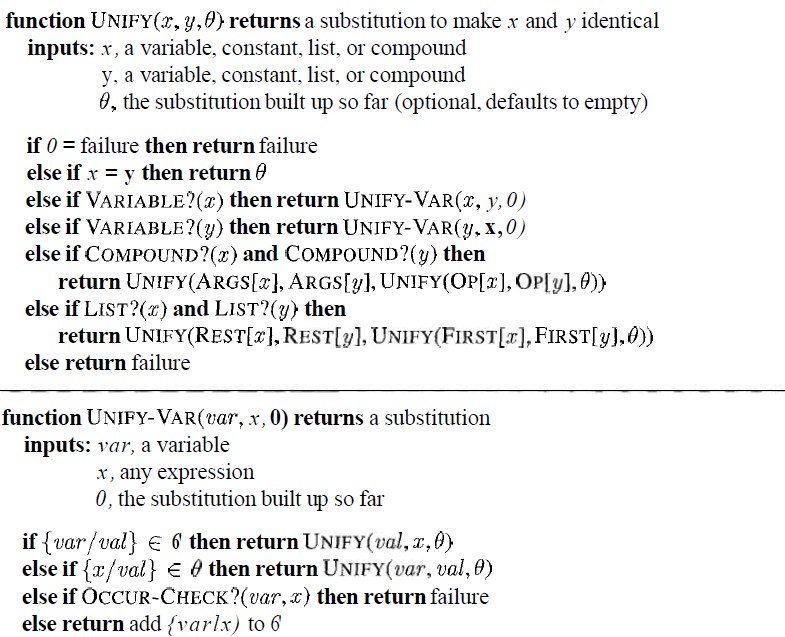
Still means the same. This is called **standardizing apart.**

* sometimes it is possible for more than one unifier returned:

##### UNIFY (Knows (John, x), Knows(y, z)) =???

This can return two possible unifications: {y/ John, x/ z} which means Knows (John, z) OR {y/ John, x/ John, z/ John}. For each unifiable pair of expressions there is a single **most general unifier (MGU)**, In this case it is *{y/ John, x/z)*.

An algorithm for computing most general unifiers is shown below.



The process is very simple: recursively explore the two expressions simultaneously "side by side," building up a unifier along the way, but failing if two corresponding points in the structures do not match. **Occur check** step makes sure same variable isn’t used twice.

***Figure 2.1*** *The unification algorithm. The algorithm works by comparing the structures of the inputs, element by element. The substitution 0 that is the argument to UNIFY is built up along the way and is used to make sure that later comparisons are consistent with bindings that were established earlier. In a compound expression, such as F (A, B), the function OP picks out the function symbol F and the function ARCS picks out the argument list (A, B).*

##### Storage and retrieval

* STORE(s) stores a sentence *s* into the knowledge base
* FETCH(s) returns all unifiers such that the query q unifies with some sentence in the knowledge base.

Easy way to implement these functions is Store all sentences in a long list, browse list one sentence at a time with UNIFY on an ASK query. But this is inefficient.

To make FETCH more efficient by ensuring that unifications are attempted only with sentences that have *some* chance of unifying. (i.e. Knows(John, x) vs. Brother(Richard, John) are not compatible for unification)

* To avoid this, a simple scheme called ***predicate indexing***puts all the *Knows* facts in one bucket and all the *Brother* facts in another.
* The buckets can be stored in a hash table for efficient access. Predicate indexing is useful when there are many predicate symbols but only a few clauses for each symbol.

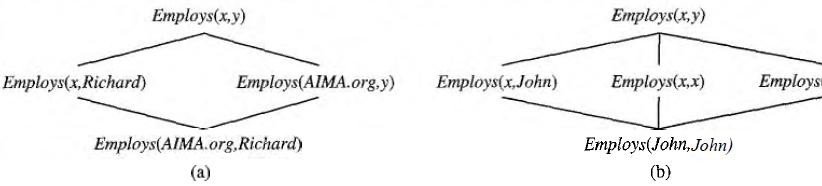
But if we have many clauses for a given predicate symbol, facts can be stored under multiple index keys.

For the fact *Employs (AIMA.org, Richard),* the queries are *Employs (A IMA. org, Richard)* Does AIMA.org employ Richard? *Employs (x, Richard)* who employs Richard?

*Employs (AIMA.org, y)* whom does AIMA.org employ?

*Employs Y(x),* who employs whom?

We can arrange this into a **subsumption lattice,** as shown below**.**

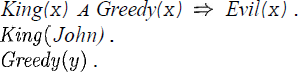


A subsumption lattice has the following properties:

***Figure 2.2*** *(a) The subsumption lattice whose lowest node is the sentence Employs (AIMA.org, Richard). (b) The subsumption lattice for the sentence Employs (John, John).*

* child of any node obtained from its parents by one substitution
* the ―highest‖ common descendant of any two nodes is the result of applying their most general unifier
* predicate with n arguments contains O(2n ) nodes (in our example, we have two arguments, so our lattice has four nodes)
* Repeated constants = slightly different lattice.
  1. ***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":

**---------** (1)

"Nono . . . has some missiles." The sentence 3 x *Owns (Nono,* .rc) *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 lo 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.

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) an*d *Weapon (M1)* is added.

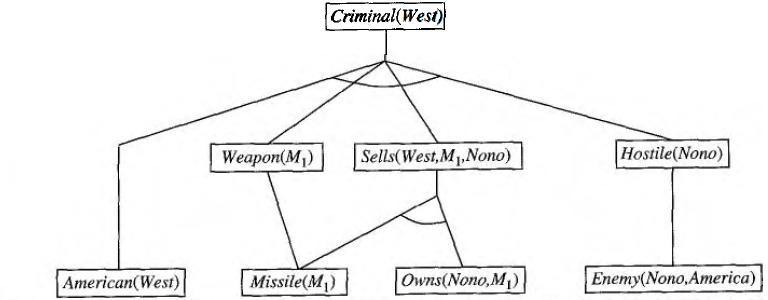
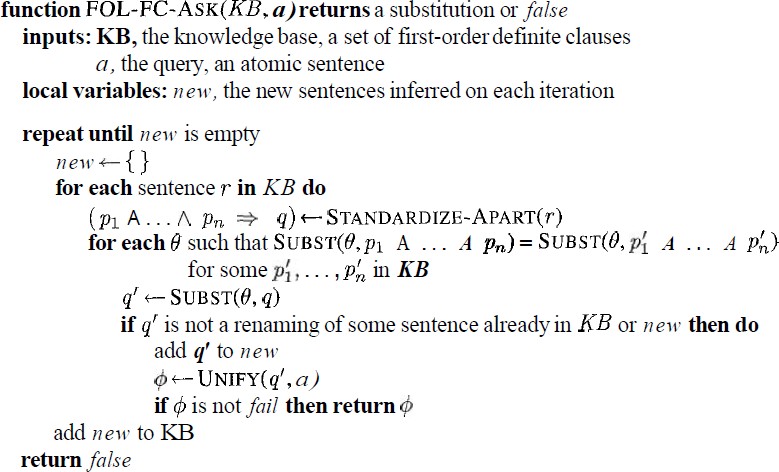
Rule (6) is satisfied with *{x/Nono},* and *Hostile (Nono)* is added.

* 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

***Figure 3.1 A*** *conceptually straightforward, but very inefficient, forward-chaining algorithm. On each iteration, it adds to KB all the atomic sentences that can be inferred in one step from the implication sentences and the atomic sentences already in KB.*



###### Efficient forward chaining:

***Figure 3.2*** *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.*

The above given forward chaining algorithm was lack with efficiency due to the the three sources of complexities:

* Pattern Matching
* Rechecking of every rule on every iteration even a few additions are made to rules
* Irrelevant facts

###### Matching rules against known facts:

For example, consider this rule,

##### Missile(x) A Owns (Nono, x) =>Sells (West, x, Nono).

The algorithm will check all the objects owned by Nono in and then for each object, it could check whether it is a missile. This is the ***conjunct ordering problem:***

―Find an ordering to solve the conjuncts of the rule premise so that the total cost is minimized‖. The **most constrained variable** heuristic used for CSPs would suggest ordering the conjuncts to look for missiles first if there are fewer missiles than objects that are owned by Nono.

The connection between pattern matching and constraint satisfaction is actually very close. We can view each conjunct as a constraint on the variables that it contains-for example, Missile(x) is a unary constraint on x. Extending this idea, we can express everyfinite-domain CSP as a single definite clause together with some associated ground facts. Matching a definite clause against a set of facts is NP-hard

1. ***Incremental forward chaining:***

On the second iteration, the rule ***Missile (x) =>Weapon (x)***

Matches against Missile (M1) (again), and of course the conclusion Weapon(x/M1) is already known so nothing happens. Such redundant rule matching can be avoided if we make the following observation:

―Every new fact inferred on iteration t must be derived from at leastone new fact inferred on iteration t – 1‖.

This observation leads naturally to an incremental forward chaining algorithm where, at iteration t, we check a rule only if its premise includes a conjunct p, that unifies with a fact p: newly inferred at iteration t - 1. The rule matching step then fixes p, to match with p’, but allows the other conjuncts of the rule to match with facts from any previous iteration.

###### Irrelevant facts:

* One way to avoid drawing irrelevant conclusions is to use backward chaining.
* Another solution is to restrict forward chaining to a selected subset of rules
* A third approach, is to rewrite the rule set, using information from the goal.so that only relevant variable bindings-those belonging to a so-called **magic** set-are considered during forward inference.

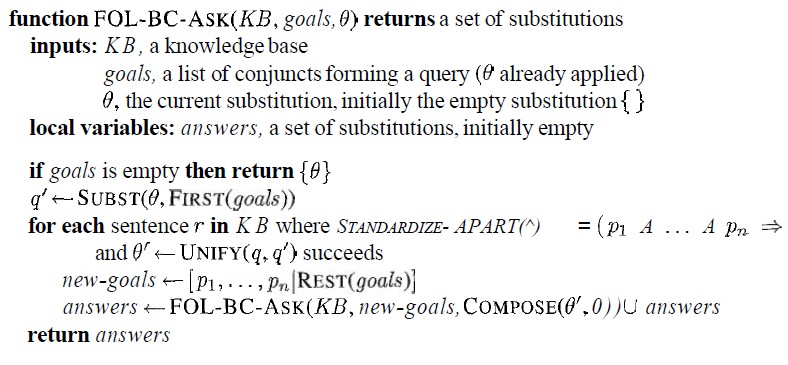
For example, if the goal is Criminal (West), the rule that concludes Criminal (x) will be rewritten to include an extra conjunct that constrains the value of x:

##### Magic(x) A American(z) A Weapon(y)A Sells(x, y, z) A Hostile(z) =>Criminal(x )

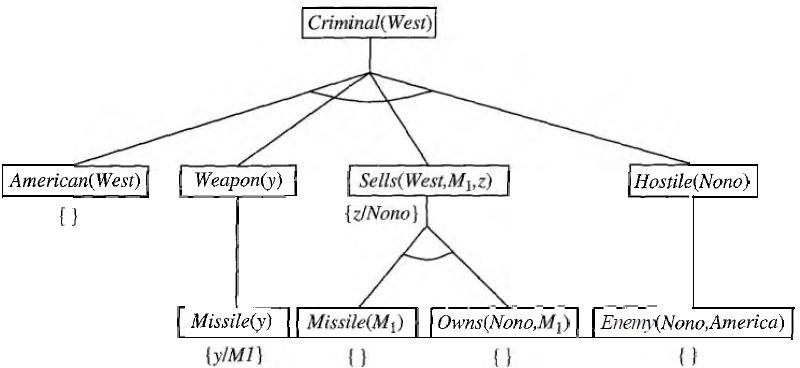
The fact *Magic (West)* is also added to the KB. In this way, even if the knowledge base contains facts about millions of Americans, only Colonel West will be considered during the forward inference process.

1. ***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.

.

***Figure 4.1****A simple backward-chaining algorithm.*



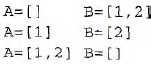
###### Logic programming:

***Figure 4.2*** *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.*

* **Prolog** is by far the most widely used logic programming language.
* Prolog programs are sets of definite clauses written in a notation different from standard first-order logic.
* Prolog uses uppercase letters for variables and lowercase for constants.
* Clauses are written with the head preceding the body; " : -" is used for left implication, commas separate literals in the body, and a period marks the end of a sentence



Prolog includes "syntactic sugar" for list notation and arithmetic. Prolog program for append (X, Y, Z), which succeeds if list Z is the result of appending lists x and Y

For example, we can ask the query append (A, B, [1, 2]): what two lists can be appended to give [1, 2]? We get back the solutions

* The execution of Prolog programs is done via depth-first backward chaining
* Prolog allows a form of negation called **negation as failure.** A negated goal not P is considered proved if the system fails to prove p. Thus, the sentence

**Alive (X) : - not dead(X)** can be read as "Everyone is alive if not provably dead."

* Prolog has an equality operator, =, but it lacks the full power of logical equality. An equality goal succeeds if the two terms are *unifiable* and fails otherwise. So X+Y=2+3 succeeds with x bound to *2* and Y bound to 3, but Morningstar=evening star fails.
* The occur check is omitted from Prolog's unification algorithm.

###### Efficient implementation of logic programs:

The execution of a Prolog program can happen in two modes: interpreted and compiled.

* Interpretation essentially amounts to running the FOL-BC-ASK algorithm, with the program as the knowledge base. These are designed to maximize speed.

First, instead of constructing the list of all possible answers for each sub goal before continuing to the next, Prolog interpreters generate one answer and a "promise" to generate the rest when the current answer has been fully explored. This promise is called a **choice point.**FOL-BC-ASK spends a good deal of time in generating and composing substitutions

when a path in search fails. Prolog will backup to previous choice point and unbind some variables. This is called ―TRAIL‖. So, new variable is bound by UNIFY-VAR and it is pushed on to trail.

* Prolog Compilers compile into an intermediate language i.e., Warren Abstract Machine or WAM named after David. H. D. Warren who is one of the implementers of first prolog compiler. So, WAM is an abstract instruction set that is suitable for prolog and can be either translated or interpreted into machine language.

**Continuations are used** to implement choice point’scontinuation as packaging up a procedure and a list of arguments that together define what should be done next whenever the current goal succeeds.

* Parallelization can also provide substantial speedup. There are two principal sources of parallelism

1. The first, called **OR-parallelism,** comes from the possibility of a goal unifying with many different clauses in the knowledge base. Each gives rise to an independent branch in the search space that can lead to a potential solution, and all such branches can be solved in parallel.
2. The second, called **AND-parallelism,** comes from the possibility of solving each conjunct in the body of an implication in parallel. AND-parallelism is more difficult to achieve, because solutions for the whole conjunction require consistent bindings for all the variables.

###### Redundant inference and infinite loops:

Consider the following logic program that decides if a path exists between two points on a directed graph.

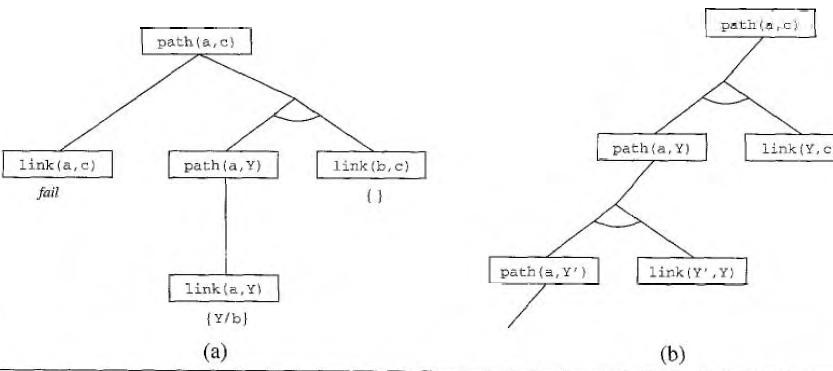


A simple three-node graph, described by the facts link (a, b) and link (b, c)



It generates the query path (a, c)

Hence each node is connected to two random successors in the next layer.



###### Constraint logic programming:

*Figure 4.3 (a) Proof that a path exists from A to C. (b) Infinite proof tree generated when the clauses are in the "wrong" order.*

The Constraint Satisfaction problem can be solved in prolog as same like backtracking algorithm.

Because it works only for finite domain CSP’s in prolog terms there must be finite number of solutions for any goal with unbound variables.



* If we have a query, triangle (3, 4, and 5) works fine but the query like, triangle (3, 4, Z) no solution.
* The difficulty is variable in prolog can be in one of two states i.e., Unbound or bound.
* Binding a variable to a particular term can be viewed as an extreme form of constraint namely ―equality‖.CLP allows variables to be constrained rather than bound.

The solution to triangle (3, 4, Z) is Constraint 7>=Z>=1.

***5. Resolution***

As in the propositional case, first-order resolution requires that sentences be in **conjunctive normal form** (CNF) that is, a conjunction of clauses, where each clause is a disjunction ofliterals.

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. We will illustrate the procedure by translating the sentence

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



The steps are as follows:

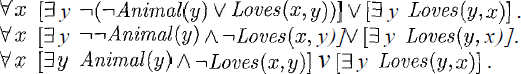
* Eliminate implications:



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



Our sentence goes through the following transformations:



* Standardize variables: For sentences like  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



* + Skolemize: Skolemization is the process of removing existential quantifiers by elimination. Translate *3 x P(x)* into *P(A),* where *A* is a new constant. If we apply this rule to our sample sentence, however, we obtain



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:*



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.

* + 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
  + Distribute V over A



This is the CNF form of given sentence.

###### The resolution inference rule:

The resolution rule for first-order clauses is simply a lifted version of the propositional resolution rule. Propositional literals are complementary if one is the negation of the other; first-order literals are complementary if one ***unifies with*** the negation of the other. Thus we have



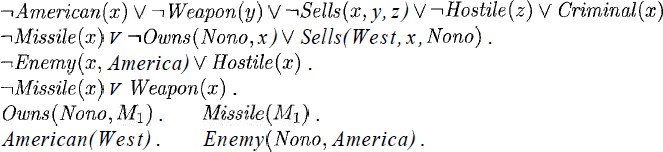
Where UNIFY (li, m j*)* == θ*.*

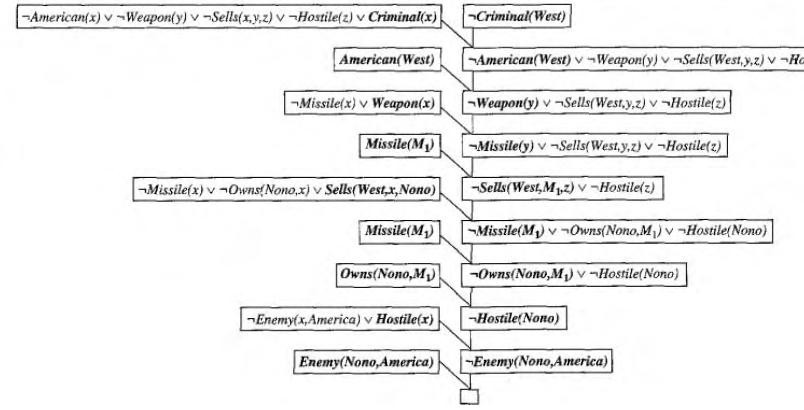
For example, we can resolve the two clauses



By eliminating the complementary literals *Loves (G(x), x)* and ¬*Loves (u, v),* with unifier θ = *{u/G(x), v/x),* to produce the resolvent clause

###### Example proofs:

Resolution proves that KB /= *a* by proving *KB* A *la* unsatisfiable, i.e., by deriving the empty clause. The sentences in CNF are

The resolution proof is shown in below figure;

Notice the structure: single "spine" beginning with the goal clause, resolving against clauses from the knowledge base until the empty clause is generated. Backward chaining is really just a

***Figure 5.1 A resolution proof that West is a criminal.***

special case of resolution with a particular control strategy to decide which resolution