UNIT 4 FIRST-ORDER LOGIC

4.1 REPRESENTATION REVISITED:

Programming languages (such as C++ or Java or Lisp) are by far the largest class of formal languages in common use.

This procedural approach can be contrasted with the declarative nature of propositional logic, in which knowledge and inference are separate, and inference is entirely domain-independent.

Programs they lack the expressiveness required to handle partial information.

- Propositional logic has sufficient expressive power to deal with partial information, using disjunction and negation.
- Propositional logic has a third 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, "S1,4 ^ S1,2" is related to the meanings of "S1,4" and "S1,2".

- Propositional logic lacks the expressive power to describe an environment with many objects concisely.
- For example, we were forced to write a separate rule about breezes and pits for each square, such as

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

In English, on the other hand, it seems easy enough to say, once and for all," Squares adjacent to pits are breezy".

Natural languages are also noncompositional --the meaning of a sentence such as "Then she saw it" can depend on a context constructed by many preceeding and succeeding sentences.

Finally, natural languages suffer from ambiguity, which would cause difficulties for thinking.
When we look at the syntax of natural language, the most obvious elements are:

- ✓ Nouns and noun phrases that refer to objects (squares , pits, wumpuses)
- ✓ Verbs and verb phrases that refer to relations among objects(is breezy, is adjacent to, shoots).
- ✓ Some of these relations are functions –relations in which there is only one "value" for a given "input". It is easy to start listing examples of objects, relations and functions:

Objects: People ,houses, numbers, theories, Ronald McDonald ,colors ,baseball games, wars, centuries….

Relations: these can be unary relations or properties such as red, round, bogus, prime, multistoried ……, or more general n-ary relations such as brother of, bigger than, inside, part of, has color, occurred after, owns, comes between, ……

Functions: father of, best friend, third inning of, more than, beginning of . . . . .

- ❖ The language of first-order logic, whose syntax and semantics we will define in the next section, is built around objects and relations.
- ❖ First-order logic can also express facts about some or all of the objects in the universe.
- ❖ This enables one to represent general laws or rules, such as the statement "Squares neighboring the Wumpus are smelly".

- ➢ The primary difference between propositional and first-order logic lies in the ontological commitment made by each language—that is, what it assumes about the nature of reality.

❖ For example, propositional logic assumes that there are facts that either hold or do not hold in the world. Each fact can be in one of two states: true or false.

❖ Temporal logic assumes that facts hold at particular times and that those times (which may be points or intervals) are ordered.

Higher-order logic views the relations and functions referred to by first-order logic as objects in themselves. This allows one to make assertions about object all relations. A student of AI must develop a talent for working with logical notation.

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.

4.2 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 .
- Models for first-order logic are more interesting.
- The domain of a model is the set of objects it contains ; these objects are sometimes called domain elements.
- figure 8.2 shows a model with five objects: The objects in the model may be related in various ways.
- In the figure, Richard and john are brothers. Formally speaking , a relation is just the set of tuples of objects that are related.(A tuple is a collection of objects arranged in a fixed order and is written with angle brackets surrounding the object relation).

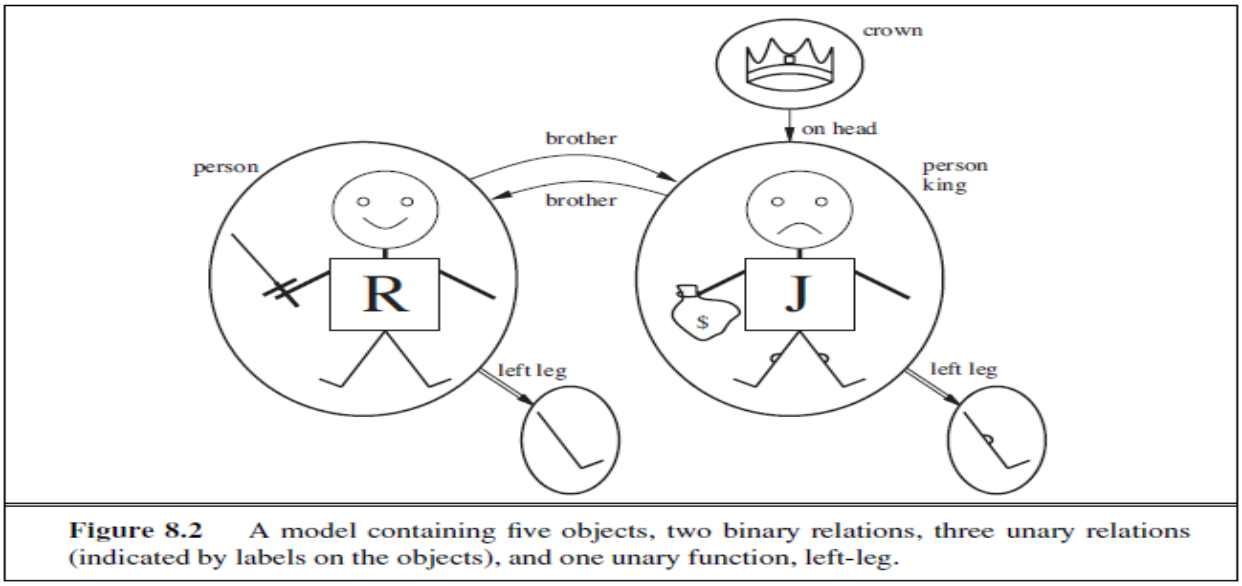


Figure 8.2 A model containing five objects, two binary relations, three unary relations (indicated by labels on the objects), and one unary function, left-leg.

The crown is on king John's head, so the " on head" relation contains just one tuple,{ the crown, king john}.

The "brother" and "on head" relations are binary relations—that is, they relate pairs of objects.

The model also contains unary relations , or properties:

- ✓ The "person" property is true of both Richard and John;
- ✓ the "king" property is true only of john(presumably because Richard is dead at this point); and
- ✓ the "crown" property is true only of the crown.

SYMBOLS AND INTERPRETATIONS:

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

➢ 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.

One possible interpretations for our example – which we will call the intended L

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

● Brother refers to the brotherhood relation, that is , the set of tuples of objects given in equation(8.1); 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 , that is, the mapping given in Equation(8.2).

➢ There are five objects in the model, so there are 25 possible interpretations just for the constant symbols Richard and John. Notice that not all the objects need have a name.

- for example, the intended interpretation does not name the crown or the legs. It is also possible for an object to have several names; there is an interpretation under which both Richard and John refer to the crown.
- If you find this possibility confusing, remember that , in propositional logic, it is perfectly possible to have a model in which Cloudly and Sunny are both true ; it is the job of the knowledge base to rule out models that are inconsistent with our knowledge.
- It is important to note that the number of domain elements in each model may be unbounded—
- for example, the domain elements may be integers or real numbers. With the symbols in our example there roughly 1025 combinations for a domain with five objects .

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

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

TERMS:

- A term is a logical expression that refers to an object. Constant symbols are therefore terms, but it is not always convenient to have a distinct symbol to name every object.
  - For example, in English we might use the expression “King John’s left leg” rather than giving a name to his leg. This is what function symbols are for: instead of using a constant symbol, we use LeftLeg(john).
  - In general case ,a complex term is formed by a function symbol followed by a parenthesized list of terms as arguments to the function symbol. It is not a “subroutine call” that “returns a value”.
  - The formal semantics of terms is straight forward.

consider a term f(t1…….tn).The function symbol f refers to some function in the model(call if F);the argument terms refer to objects in the domain.

- **ATOMIC SENTENCES:**

An atomic sentence is formed from a predicate symbol followed by a parenthesized list of terms:

Brother(Richard,John).

Richard the Lionheart is the brother of King John. Atomic sentences can have complex terms as arguments. Thus,

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

States that Richard the Lionheart’s father is married to King John’s mother.

- **COMPLEX SENTENCES:**
  We can use logical connectives to construct more complex sentences, just as in propositional calculus.

  ¬Brother (LeftLeg(Richard), John)
  Brother (Richard, John) ∧ Brother (John, Richard)
  King(Richard) ∨ King(John)
  ¬King(Richard) ⇒ King(John)

QUANTIFIERS:
First-order logic contains two standard quantifiers called universal and existential.

• UNIVERSAL QUANTIFIERS:
"All kings are persons" , is written in first-order logic as :

$$\forall\ x\ \text{King}(x) \Rightarrow \text{Person}(x)\ .$$

∀ is usually pronounced "For all ...". (Remember that the upside-down A stands for "all.") Thus, the sentence says, "For all x, if x is a king, then x is a person."
The symbol x is called VARIABLE 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 GROUND TERM with no variables is called a ground term.

Intuitively, the sentence ∀ x P, where P is any logical expression, says that P is true for every object x.
More precisely, ∀ x P is true in a given model if P is true in all possible extended interpretations constructed from the interpretation given in the model.The extended interpretations in five ways are illustrated below:

- ❖ 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

We see that the implication is true whenever its premise is false—regardless of the truth of the conclusion.

- **EXISTENTIAL QUANTIFIER:**

  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. for example, that King John has a crown on his head, we write

$$\exists x\ \text{Crown}(x) \wedge \text{OnHead}(x, \text{John})\ .$$

∃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 if P is true in at least one extended interpretation that assigns x to a domain element. That is, at least one of the following is true:

- ❖ Richard the Lionheart is a crown ∧ Richard the Lionheart is on John's head;
- ❖ King John is a crown ∧ King John is on John's head;
- ❖ Richard's left leg is a crown ∧ Richard's left leg is on John's head;
- ❖ John's left leg is a crown ∧ John's left leg is on John's head;
- ❖ The crown is a crown ∧ the crown is on John's head.

This is entirely consistent with the original sentence "King John has a crown on his head."

- ➢ Just as ⇒ appears to be the natural connective to use with ∀, ∧ is the natural connective to use with ∃.
- ➢ Using ∧ as the main connective with ∀ led to an overly strong statement in the example in the previous section;
- ➢ using ⇒ with ∃ usually leads to a very weak statement, indeed.

Consider the following sentence:

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

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 ⇒ Richard the Lionheart 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;

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

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

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

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

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

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

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

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

- ❖ The order of quantification is therefore very important. It becomes clearer if we insert parentheses. ∀ x (∃ y Loves(x, y)) says that everyone has a particular property, namely, the property that they love someone.
- ❖ On the other hand, ∃ y (∀ x Loves(x, y)) says that someone in the world has a particular property, namely the property of being loved by everybody.

- ❖ Some confusion can arise when two quantifiers are used with the same variable name. Consider the sentence

$$\forall\ x\ (\text{Crown}(x) \vee (\exists\ x\ \text{Brother}\ (\text{Richard}, x)))\ .$$

- ❖ Here the x in Brother (Richard, x) is existentially quantified.
- ❖ The rule is that the variable belongs to the innermost quantifier that mentions it: it will not be subject to any other quantification.
- ❖ Another way to think of it is this:

∃ x Brother (Richard, x) is a sentence about Richard (that he has a brother), not about x; so putting a ∀ x outside it has no effect. It could equally well have been written ∃ z Brother (Richard, z).

Because this can be a source of confusion, we will always use different variable names with nested quantifiers.

- **Connections between ∀ and ∃:**

The two quantifiers are actually intimately connected with each other, through negation. Asserting that everyone dislikes parsnips is the same as asserting there does not exist someone who likes them, and vice versa:

∀ x ¬Likes(x,Parsnips) is equivalent to ¬∃x Likes(x,Parsnips) .

We can go one step further: "Everyone likes ice cream" means that there is no one who does not like ice cream:

∀ x Likes(x,IceCream) is equivalent to ¬∃ x ¬Likes(x,IceCream) .

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

- ∀ x ¬P ≡ ¬∃ x P ¬(P ∨ Q) ≡ ¬P ∧ ¬Q
- ¬∀ x P ≡ ∃ x ¬P ¬(P ∧ Q) ≡ ¬P ∨ ¬Q
- ∀ x P ≡ ¬∃ x ¬P P ∧ Q ≡ ¬(¬P ∨ ¬Q)
- ∃ x P ≡ ¬∀ x ¬P P ∨ Q ≡ ¬(¬P ∧ ¬Q) .

Thus, we do not really need both ∀ and ∃, just as we do not really need both ∧ and ∨ . Still, readability is more important than parsimony, so we will keep both of the quantifiers.

- **Equality:**

First-order logic includes one more way to make atomic sentences, other than using a predicate and terms as described earlier. 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.

- The equality symbol can be used to state facts about a given function, as we just did for the Father symbol.

It can also be used with negation to insist that two terms are not the same object. To say that Richard has at least two brothers, we would write

∃ x, y Brother (x, Richard) ∧ Brother (y, Richard) ∧ ¬(x = y) .

4.3 USING FIRST ORDER LOGIC:

- We begin with a brief description of the TELL/ASK interface for first-order knowledge bases.
- We can ask questions of the knowledge base using ASK.
  - For example:ASK(KB, King(John))
- QUERY returns true. Questions asked with ASK are called queries or goals.
  - ASK(KB, Person(John))
- should also return true. We can ask quantified queries, such as ASK(KB, ∃ x Person(x)) .

- The answer is true, is asking"Is there an x such that……..,".
- The standard form for an answer of this sort is a **Substitution** or **Binding List,** the answer would be {x/John}.

The kinship domain:

- we consider is the domain of family relationships, two unary predicates, Male and Female.
- Kinship relations—parenthood, brotherhood, marriage, and so on—are represented by binary predicates: We use functions for Mother and Father , because every person has exactly one of each of these .

For example:

1) **one's mother is one's female parent:**

   ∀ m, c Mother (c) = m ⇔ Female(m) ∧ Parent(m, c) .

2) **One's husband is one's male spouse:**
   ∀ w, h Husband(h, w) ⇔ Male(h) ∧ Spouse(h, w) .
3) **Male and female are disjoint categories:**
   ∀ x Male(x) ⇔ ¬Female(x) .
4) **Parent and child are inverse relations:**
   ∀ p, c Parent(p, c) ⇔ Child(c, p) .
5) **A grandparent is a parent of one's parent:**
   ∀ g, c Grandparent(g, c) ⇔ ∃ p Parent(g, p) ∧ Parent(p, c) .
6) **A sibling is another child of one's parents:**
   ∀ x, y Sibling(x, y) ⇔ x = y ∧ ∃ p Parent(p, x) ∧ Parent(p, y) .

- Each of these sentences can be viewed as an axiom of the kinship domain.
- Axioms are commonly associated with purely mathematical domains. Our kinship axioms are also definitions.
- Not all logical sentences about a domain are axioms. Some are theorems
- For example: consider the assertion that siblinghood is symmetric:
    - ∀ x, y Sibling(x, y) ⇔ Sibling(y, x) .
- In fact, it is a theorem that follows logically from the axiom that defines siblinghood.
- From a practical point of view, theorems are essential to reduce the computational cost of deriving new sentences.
- Not all axioms are definitions. Some provide more general information about certain predicates without constituting a definition.
- For example:
- there is no obvious definitive way to complete the sentence :
    - ∀ x Person(x) ⇔ ...
    - ∀ x Person(x) ⇒ ... ∀ x ... ⇒ Person(x) .
- Axioms can also be "just plain facts," such as Male(Jim) and Spouse(Jim, Laura).

Numbers, sets, and lists

- Numbers are perhaps the most vivid example of how a large theory can be built up from a tiny kernel of axioms.
- We describe here the theory of natural numbers or non-negative integers. We need a predicate NatNum that will be true of natural numbers
- we need one constant symbol, 0; and we need one function symbol, S (successor).
    - **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) ∧ NatNum(n) ⇒ + (S(m), n) = S(+(m, n)) .**
- The use of infix notation is an example of syntactic sugar, that is, an extension to or abbreviation of the standard syntax that does not change the semantics.
- We will use the normal vocabulary of set theory as syntactic sugar. 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 and s1 ⊆ s2 The binary functions are s1 ∩ s2, s1 ∪ s2, and {x|s}

One possible set of axioms is as follows:

1) **The only sets are the empty set and those made by adjoining something to a set:**
   ∀ s Set(s) ⇔ (s = { }) ∨ (∃ x, s2 Set(s2) ∧ s = {x|s2}) .
2) **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} = { } .
3) **Adjoining an element already in the set has no effect:**
   ∀ x, s x∈ s ⇔ s = {x|s} .
4) **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} ∧ (x = y ∨ x∈ s2)) .
5) **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) .

6) **Two sets are equal if and only if each is a subset of the other:**
   ∀ s1, s2 (s1 = s2) ⇔ (s1 ⊆ s2 ∧ s2 ⊆ s1) .
7) **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 ∧ x∈s2) .
8) **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 ∪ s2) ⇔ (x∈ s1 ∨ x∈s2) .

Lists

Lists are similar to sets. The differences are that lists are ordered and the same element can appear more than once in a list.

We can use the vocabulary of Lisp for lists:

Consider the wumpus world problem,

- 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}.
- BestAction(Grab, 5)—that is, Grab is the right thing to do.

Adjacency of any two squares can be defined as

$\forall x, y, a, b \; Adjacent([x, y], [a, b]) \Leftrightarrow (x = a \wedge (y = b - 1 \vee y = b + 1)) \vee (y = b \wedge (x = a - 1 \vee x = a + 1))$.

The agent's location changes over time, so we write At(Agent, s, t) to mean that the agent is at square s at time t. We can fix the wumpus's location with ∀t At .

We can then say that objects can only be at one location at a time:

$$\forall x, s1, s2, t \; At(x, s1, t) \wedge At(x, s2, t) \Rightarrow s1 = s2 .$$

$$\forall s \; Breezy(s) \Leftrightarrow \exists r \; Adjacent(r, s) \wedge Pit(r) .$$

4.4 Knowledge engineering in first order logic:

- This section describes the general process of knowledge-base construction— a process called knowledge engineering.
- We illustrate the knowledge engineering process in an electronic circuit domain that should already be fairly familiar, so that we can concentrate on the representational issues involved.

The knowledge-engineering process

Knowledge engineering projects vary widely in content, scope, and difficulty, but all such projects include the following steps:

1. **Identify the task.**
   The knowledge engineer must delineate the range of questions that the knowledge base will support and the kinds of facts that will be available for each specific problem instance.
   The task will determine what knowledge must be represented in order to connect problem instances to answers.

2. **Assemble the relevant knowledge**

The knowledge engineer might already be an expert in the domain, or might need to work with real experts to extract what they know.

The idea is to understand the scope of the knowledge base, as determined by the task, and to understand how the domain actually works.

3. **Decide on a vocabulary of predicates, functions, and constants.**

That is, translate the important domain-level concepts into logic-level names.

This can have a significant impact on the eventual success of the project.

For example,

Should pits be represented by objects or by a unary predicate on squares?

Should the agent's orientation be a function or a predicate?

Should the wumpus's location depend on time?

4. **Encode general knowledge about the domain.**

The knowledge engineer writes down the axioms for all the vocabulary terms.

This pins down the meaning of the terms.

It will involve writing simple atomic sentences about instances of concepts.

For example, if an axiom is missing, some queries will not be answerable from the knowledge base.

Missing axioms can be easily identified by noticing places where the chain of reasoning stops unexpectedly.

The electronic circuit domain

1. **Identify the task**

- At the highest level, one analyzes the circuit's functionality.
- Q1: If all the inputs are high, what is the output of gate A2?
- Q2: What are all the gates connected to the first input terminal?
- Q3: Does the circuit contain feedback loops?
- There are more detailed levels of analysis, including those related to timing delays, circuit area, power consumption, production cost, and so on.

2. **Assemble the relevant knowledge** :

- What do we know about digital circuits? For our purposes, they are composed of wires and gates. Signals flow along wires to the input terminals of gates.
- For resolving timing faults, we would need to include gate delays.
- If we were interested in designing a product that would be profitable, then the cost of the circuit and its speed relative to other products on the market would be important.

3. **Decide on a vocabulary** :

- First, we need to be able to distinguish gates from each other and from other objects.
- Type(X1) = XOR.
- Circuits, like gates, are identified by a predicate: Circuit(C1). Next we consider terminals, which are identified by the predicate Terminal(x).
- We use the function In(1, X1) to denote the first input terminal for gate X1.
- A similar function Out is used for output terminals. The function Arity(c, i, j) says that circuit c has i input and j output terminals.
- The connectivity between gates can be represented by a predicate, Connected, which takes two terminals as arguments, as in Connected(Out(1, X1),In(1, X2)).

- Finally, we need to know whether a signal is on or off. One possibility is to use a unary predicate, On(t), which is true when the signal at a terminal is on.
- And a function Signal(t) that denotes the signal value for the terminal t.

4. <u>Encode general knowledge of the domain</u>

One sign that we have a good ontology is that we require only a few general rules, which can be stated clearly and concisely. These are all the axioms we will need:

1. If two terminals are connected, then they have the same signal:

$\forall$ t1, t2 Terminal(t1) $\wedge$ Terminal(t2) $\wedge$ Connected(t1, t2) $\Rightarrow$ Signal(t1) = Signal(t2) .

2. The signal at every terminal is either 1 or 0:
$$\forall t \;\; Terminal(t) \Rightarrow Signal(t) = 1 \vee Signal(t) = 0 \;\; .$$
3. Connected is commutative:
$$\forall t_1, t_2 \;\; Connected(t_1, t_2) \Leftrightarrow Connected(t_2, t_1) \; .$$
4. There are four types of gates:
$$\forall g \;\; Gate(g) \wedge k = Type(g) \Rightarrow k = AND \vee k = OR \vee k = XOR \vee k = NOT \; .$$
5. An AND gate's output is 0 if and only if any of its inputs is 0:
$$\forall g \;\; Gate(g) \wedge Type(g) = AND \Rightarrow Signal(Out(1, g)) = 0 \Leftrightarrow \exists n \;\; Signal(In(n, g)) = 0 \; .$$
6. An OR gate's output is 1 if and only if any of its inputs is 1:
$$\forall g \;\; Gate(g) \wedge Type(g) = OR \Rightarrow Signal(Out(1, g)) = 1 \Leftrightarrow \exists n \;\; Signal(In(n, g)) = 1 \; .$$
7. An XOR gate's output is 1 if and only if its inputs are different:
$$\forall g \;\; Gate(g) \wedge Type(g) = XOR \Rightarrow Signal(Out(1, g)) = 1 \Leftrightarrow Signal(In(1, g)) \neq Signal(In(2, g)) \; .$$
8. A NOT gate's output is different from its input:
$$\forall g \;\; Gate(g) \wedge (Type(g) = NOT) \Rightarrow Signal(Out(1, g)) \neq Signal(In(1, g)) \; .$$
9. The gates (except for NOT) have two inputs and one output.
$$\forall g \;\; Gate(g) \wedge Type(g) = NOT \Rightarrow Arity(g, 1, 1) \;\; .$$
$$\forall g \;\; Gate(g) \wedge k = Type(g) \wedge (k = AND \vee k = OR \vee k = XOR) \Rightarrow Arity(g, 2, 1)$$
10. A circuit has terminals, up to its input and output arity, and nothing beyond its arity:
$$\forall c, i, j \;\; Circuit(c) \wedge Arity(c, i, j) \Rightarrow \forall n \;\; (n \leq i \Rightarrow Terminal(In(c, n))) \wedge (n > i \Rightarrow In(c, n) = Nothing) \wedge \forall n \;\; (n \leq j \Rightarrow Terminal(Out(c, n))) \wedge (n > j \Rightarrow Out(c, n) = Nothing)$$
11. Gates, terminals, signals, gate types, and *Nothing* are all distinct.
$$\forall g, t \;\; Gate(g) \wedge Terminal(t) \Rightarrow g \neq t \neq 1 \neq 0 \neq OR \neq AND \neq XOR \neq NOT \neq Nothing \; .$$
12. Gates are circuits.
$$\forall g \;\; Gate(g) \Rightarrow Circuit(g)$$

Encode the specific problem instance The circuit shown in Figure 8.6 is encoded as circuit C1 with the following description. First, we categorize the circuit and its component gates:

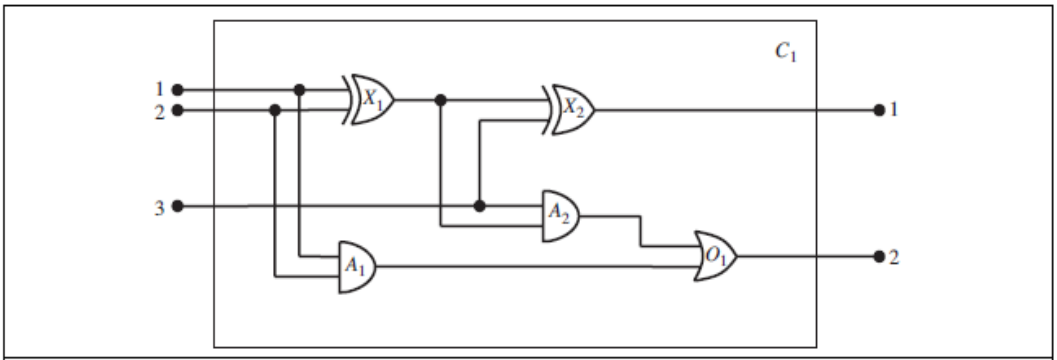


Figure 8.6 A digital circuit C1, purporting to be a one-bit full adder. The first two inputs are the two bits to be added, and the third input is a carry bit. The first output is the sum, and the second output is a carry bit for the next adder. The circuit contains two XOR gates, two AND gates, and one OR gate.

$$Circuit(C_1) \wedge Arity(C_1, 3, 2)$$
$$Gate(X_1) \wedge Type(X_1) = XOR$$
$$Gate(X_2) \wedge Type(X_2) = XOR$$
$$Gate(A_1) \wedge Type(A_1) = AND$$
$$Gate(A_2) \wedge Type(A_2) = AND$$
$$Gate(O_1) \wedge Type(O_1) = OR\,.$$

Then, we show the connections between them:

$$Connected(Out(1, X_1), In(1, X_2)) \quad Connected(In(1, C_1), In(1, X_1))$$
$$Connected(Out(1, X_1), In(2, A_2)) \quad Connected(In(1, C_1), In(1, A_1))$$
$$Connected(Out(1, A_2), In(1, O_1)) \quad Connected(In(2, C_1), In(2, X_1))$$
$$Connected(Out(1, A_1), In(2, O_1)) \quad Connected(In(2, C_1), In(2, A_1))$$
$$Connected(Out(1, X_2), Out(1, C_1)) \quad Connected(In(3, C_1), In(2, X_2))$$
$$Connected(Out(1, O_1), Out(2, C_1)) \quad Connected(In(3, C_1), In(1, A_2))\,.$$

POSE QUERIES TO THE INTERFACE PROCEDURE

- What combinations of inputs would cause the first output of C1 (the sum bit) to be 0 and the second output of C1 (the carry bit) to be 1?
  - ∃ i1, i2, i3 Signal(In(1, C1)) = i1 ∧ Signal(In(2, C1)) = i2 ∧ Signal(In(3, C1)) = i3 ∧ Signal(Out(1, C1)) = 0 ∧ Signal(Out(2, C1)) = 1 .
- The answers are substitutions for the variables i1, i2, and i3 such that the resulting sentence is entailed by the knowledge base. ASKVARS will give us three such substitutions:
  - {i1/1, i2/1, i3/0} {i1/1, i2/0, i3/1} {i1/0, i2/1, i3/1} .
- What are the possible sets of values of all the terminals for the adder circuit?
  - ∃ i1, i2, i3, o1, o2 Signal(In(1, C1)) = i1 ∧ Signal(In(2, C1)) = i2 ∧ Signal(In(3, C1)) = i3 ∧ Signal(Out(1, C1)) = o1 ∧ Signal(Out(2, C1)) = o2 .
- This is a simple example of circuit verification.

4.5 Proportional vs First-Order Inference

first-order inference can be done by converting the knowledge base to *propositional* logic and using *propositional* inference.

Inference rules for quantifiers

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

$$\forall\ x\ \text{King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)\ .$$

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

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

The rule of **Universal Instantiation** (**UI** for short) says that we can infer any sentence obtained by substituting a **ground term** (a term without variables) for the variable.[1] To write out the inference rule formally, we use the notion of **substitutions** introduced in Section 8.3. Let $\text{SUBST}(\theta, \alpha)$ denote the result of applying the substitution θ to the sentence α. Then the rule is written

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

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

In the rule for **Existential Instantiation**, the variable is replaced by a single *new constant symbol*. The formal statement is as follows: for any sentence α, 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

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

we can infer the sentence

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

as long as C_1 does not appear elsewhere in the knowledge base. Basically, the existential

<u>Reduction to propositional inference</u>

Once we have rules for inferring non quantified sentences from quantified sentences, it becomes possible to reduce first-order inference to propositional inference.

The first idea is that, just as an existentially quantified sentence can be replaced by one instantiation, a universally quantified sentence can be replaced by the set of *all possible* instantiations.

For example, suppose our knowledge base contains just the sentences

$$\forall\ x\ \text{King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$$
$$\text{King}(\text{John})$$
$$\text{Greedy}(\text{John})$$
$$\text{Brother}(\text{Richard}, \text{John})\ .$$

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

$$\text{King}(\text{John}) \wedge \text{Greedy}(\text{John}) \Rightarrow \text{Evil}(\text{John})$$
$$\text{King}(\text{Richard}) \wedge \text{Greedy}(\text{Richard}) \Rightarrow \text{Evil}(\text{Richard})\ ,$$

and 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 so on—as proposition symbols.

This technique of **propositionalization,** that entailment is preserved. Thus, we have a complete decision procedure for entailment . . . or perhaps not.

There is a problem: when the knowledge base includes a function symbol, the set of possible ground-term substitutions is infinite!
For example, if the knowledge base mentions the Father symbol, then infinitely many nested terms such as Father (Father (Father (John))) can be constructed. Our propositional algorithms will have difficulty with an infinitely large set of sentences.

4.6 WHAT IS KNOWLEDGE REPRESENTATION ?

Knowledge is the information about a domain that can be used to solve problems in that domain. To solve many problems requires much knowledge, and this knowledge must be represented in the computer. As part of designing a program to solve problems, we must define how the knowledge will be represented.

A **representation scheme** is the form of the knowledge that is used in an agent. A **representation** of some piece of knowledge is the internal representation of the knowledge. A representation scheme specifies the form of the knowledge. A **knowledge base** is the representation of all of the knowledge that is stored by an agent.This is called as **knowledge representation.**

4.6.1 ONTOLOGICAL ENGINEERING

This field of engineering describes;

How to create more general and flexible representations

- Concepts like actions, time, physical object and beliefs
- Operates on a bigger scale than K.E.

Define general framework of concepts

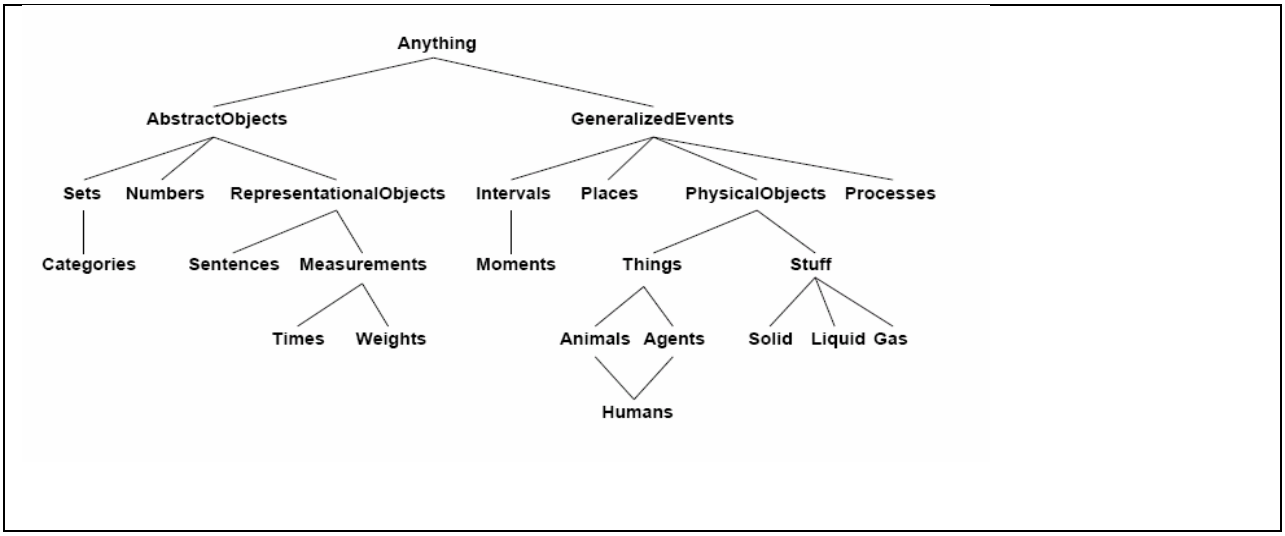
- Upper ontology

Limitations of logic representation

- Red, green and yellow tomatoes: exceptions and uncertainty.

Representing these abstract concepts is sometimes called as ontological engineering.

The upper ontology of the world



Difference with special-purpose ontologies

• A general-purpose ontology should be applicable in more or less any special-purpose domain.

- Add domain-specific axioms

• In any sufficiently demanding domain different areas of knowledge need to be unified.

- Reasoning and problem solving could involve several areas simultaneously

• What do we need to express?

- Categories, Measures, Composite objects, Time, Space, Change, Events, Processes, Physical Objects, Substances, Mental Objects, Beliefs.

4.7 Categories and objects

• KR requires the organisation of objects into categories

- Interaction at the level of the object
- Reasoning at the level of categories

• Categories play a role in predictions about objects

- Based on perceived properties

• Categories can be represented in two ways by FOL(First Order Logic)

- Predicates: apple(x)
- *Reification* of categories into objects: apples

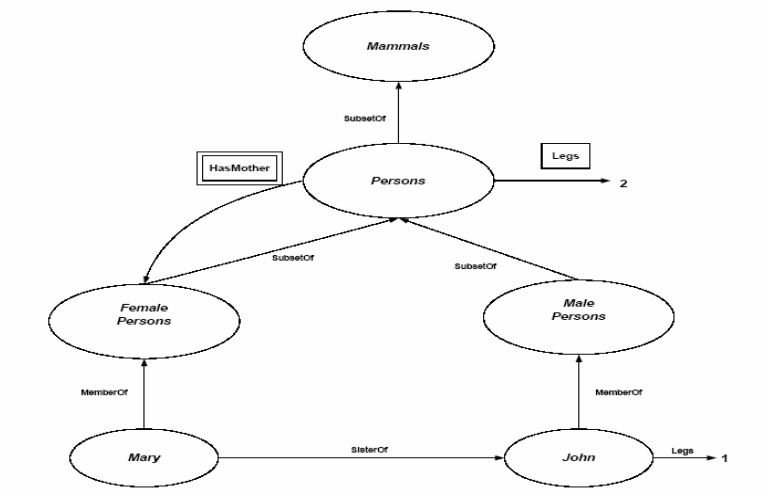
• Category = set of its members.

Category organization

• Relation = *inheritance*:

- All instance of food are edible, fruit is a subclass of food and apples is a subclass of fruit then an apple is edible.

•Defines a taxonomy



FOL and categories

• An object is a member of a category
  • MemberOf(BB12,Basketballs)
• A category is a subclass of another category
  • SubsetOf(Basketballs,Balls)
• All members of a category have some properties
  • ∀x (MemberOf(x,Basketballs) ⇒Round(x))
• All members of a category can be recognized by some properties
  • ∀x (Orange(x) ∧Round(x) ∧Diameter(x)=9.5in ∧MemberOf(x,Balls)⇒MemberOf(x,BasketBalls))
• A category as a whole has some properties
  • MemberOf(Dogs,DomesticatedSpecies)

Relations between categories

• Two or more categories are ***disjoint*** if they have no members in common:
  • Disjoint(s)⇔(∀c1,c2 c1 ∈s ∧c2 ∈s ∧c1 ≠ c2 ⇒Intersection(c1,c2) ={})
• Example: Disjoint({animals, vegetables})
• A set of categories *s* constitutes an ***exhaustive decomposition*** of a category *c* if all members of the set *c*are covered by categories in *s*:
  • E.D.(s,c) ⇔(∀i i∈c ⇒∃c2 c2∈s ∧i ∈c2)
• Example: ExhaustiveDecomposition({Americans, Canadian, Mexicans}, NorthAmericans).
• A ***partition*** is a disjoint exhaustive decomposition:
  • Partition(s,c) ⇔Disjoint(s) ∧E.D.(s,c)
• Example: Partition({Males,Females},Persons).
• Is ({Americans,Canadian, Mexicans},NorthAmericans) a partition?
• No! There might be dual citizenships.
• Categories can be defined by providing necessary and sufficient conditions for membership
  • ∀x Bachelor(x) ⇔Male(x) ∧Adult(x) ∧Unmarried(x)

Physical composition

• One object may be part of another:
  • PartOf(Bucharest,Romania)

- PartOf(Romania,EasternEurope)
- PartOf(EasternEurope,Europe)

• The PartOf predicate is transitive (and reflexive), so we can infer that PartOf(Bucharest,Europe)

• More generally:

- ∀x PartOf(x,x)
- ∀x,y,zPartOf(x,y) ∧PartOf(y,z) ⇒PartOf(x,z)

• Often characterized by structural relations among parts.

•**E.g:**

Biped(a) ⇒

$$(\exists l_1, l_2, b)(Leg(l_1) \wedge Leg(l_2) \wedge Body(b) \wedge$$
$$PartOf(l_1, a) \wedge PartOf(l_2, a) \wedge PartOf(b, a) \wedge$$
$$Attached(l_1, b) \wedge Attached(l_2, b) \wedge$$
$$l_1 \neq l_2 \wedge (\forall l_3)(Leg(l_3) \Rightarrow (l_3 = l_1 \vee l_3 = l_2)))$$

Measurements

• Objects have height, mass, cost,Values that we assign to these are **measures**

• Combine Unit functions with a number:

Length(L1) = Inches(1.5)=Centimeters(3.81).

• Conversion between units:

∀iCentimeters(2.54 x i)=Inches(i).

• Some measures have no scale: Beauty, Difficulty, etc.

- Most important aspect of measures:they are **orderable**.

• Don't care about the actual numbers.

Natural kinds

• Many categories have no clear-cut definitions (e.g., chair, bush, book).

• Tomatoes: sometimes green, red, yellow, black. Mostly round.

• One solution: subclass using category *Typical(Tomatoes).*

- Typical(c) ⊆c
- ∀x, x ∈Typical(Tomatoes) ⇒Red(x) ∧Spherical(x).

• We can write down useful facts about categories without providing exact definitions.

• Wittgenstein (1953) gives an exhaustive summary about the problems involved when exact definitions for natural kinds are required in his book "PhilosophischeUntersuchungen".

• What about "bachelor"? Quine (1953) challenged the utility of the notion of *strict definition*. We might question a statement such as "the Pope is a bachelor".

Substances and Objects

The real world can be seen as consisting of primitive objects (e.g., atomic particles) and composite objects built from them.

By reasoning at the level of large objects such as apples and cars, we can overcome the complexity involved in dealing with vast numbers of primitive objects individually.

There is, however, a significant portion of reality that seems to defy any obvious **individuation**—division into distinct objects.

$b \in Butter \wedge PartOf(p, b) \Rightarrow p \in Butter$.

We can now say that butter melts at around 30 degrees centigrade:

$b \in Butter \Rightarrow MeltingPoint(b, Centigrade(30))$

What is actually going on is this: some properties are **intrinsic**: they belong to the very substance of the object, rather than to the object as a whole.

When you cut an instance of *stuff* in half, the two pieces retain the intrinsic properties—things like density, boiling point, flavor, color, ownership, and so on.

On the other hand, their **extrinsic** properties—weight, length, shape, and so on—are not retained under subdivision.

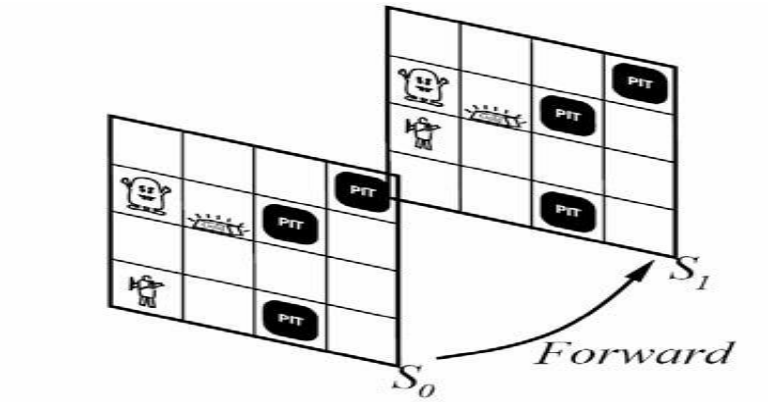
The category Stuff is the most general substance category, specifying no intrinsic properties.

The category Thing is the most general discrete object category, specifying no extrinsic properties.

4.8 Actions, events and situations

Situation calculus:

- Reasoning about outcome of actions is central to KB-agent.
- How can we keep track of location in FOL?
- Remember the multiple copies in PL.
- Representing time by situations (states resulting from the execution of actions).



- Actions are logical terms

- Situations are logical terms consiting of
    - The initial situation I
    - All situations resulting from the action on I (=*Result(a,s)*)

EVENTS:

In Section 10.4.2, we showed how situation calculus represents actions and their effects. Situation calculus is limited in its applicability: it was designed to describe a world in which actions are discrete, instantaneous, and happen one at a time.

Consider a continuous action, such as filling a bathtub. Situation calculus can say that the tub is empty before the action and full when the action is done, but it can't talk about what happens *during* the action. It also can't describe two actions happening at the same time—such as brushing one's teeth while waiting for the tub to fill. To handle such cases we introduce an alternative formalism known as **event calculus**, which is based on points of time rather than on situations.

Events are described as instances of event categories.4 The event E1 of Shankar flying from San Francisco to Washington, D.C. is described as
$E1 \in Flyings \wedge Flyer(E1, Shankar) \wedge Origin(E1, SF) \wedge Destination(E1,DC)$.

We then use Happens(E1, i) to say that the event E1 took place over the time interval i. We represent time intervals by a (start, end) pair of times; that is, i = (t1, t2).

The complete set of predicates for one version of the event calculus is

T(f, t) Fluent f is true at time t
Happens(e, i) Event e happens over the time interval i
Initiates(e, f, t) Event e causes fluent f to start to hold at time t
Terminates(e, f, t) Event e causes fluent f to cease to hold at time t
Clipped(f, i) Fluent f ceases to be true at some point during time interval i
Restored (f, i) Fluent f becomes true sometime during time interval i

We define T by saying that a fluent holds at a point in time if the fluent was initiated by an event at some time in the past and was not made false (clipped) by an intervening event.

A fluent does not hold if it was terminated by an event and not made true (restored) by another event. Formally, the axioms are:

$Happens(e, (t1, t2)) \wedge Initiates(e, f, t1) \wedge \neg Clipped(f, (t1, t)) \wedge t1 < t \Rightarrow T(f, t)$
$Happens(e, (t1, t2)) \wedge Terminates(e, f, t1) \wedge \neg Restored(f, (t1, t)) \wedge t1 < t \Rightarrow \neg T(f, t)$

where Clipped and Restored are defined by

$Clipped(f, (t1, t2)) \Leftrightarrow \exists$ e, t, t3 $Happens(e, (t, t3)) \wedge t1 \leq t < t2 \wedge Terminates(e, f, t)$
$Restored(f, (t1, t2)) \Leftrightarrow \exists e, t, t3\ Happens(e, (t, t3)) \wedge t1 \leq t < t2 \wedge Initiates(e, f, t)$

It is convenient to extend T to work over intervals as well as time points; a fluent holds over an interval if it holds on every point within the interval:

$$T(f, (t1, t2)) \Leftrightarrow [\forall\ t\ (t1 \leq t < t2) \Rightarrow T(f, t)]$$

Processes

Shankar's trip has a beginning, middle, and end. If interrupted halfway, the event would be something different. If we take a small interval of Shankar's flight, say, the third 20-minute segment, that event is still a member of Flyings. In fact, this is true for any subinterval. Categories of events with this property are called **process** categories or **liquid event** categories.

Any process e that happens over an interval also happens over any subinterval:
$(e \exists \text{ Processes}) \wedge \text{Happens}(e, (t1, t4)) \wedge (t1 < t2 < t3 < t4) \Rightarrow \text{Happens}(e, (t2, t3))$.

Time intervals

will consider two kinds of time intervals: moments and extended intervals. The distinction is that only moments have zero duration:

Partition({Moments,ExtendedIntervals}, Intervals)
$i \exists \text{Moments} \Leftrightarrow \text{Duration}(i)=\text{Seconds}(0)$.

The function Duration gives the difference between the end time and the start time.
$\text{Interval}(i) \Rightarrow \text{Duration}(i)=(\text{Time}(\text{End}(i)) - \text{Time}(\text{Begin}(i)))$.

The complete set of interval relations, as proposed by Allen (1983), is shown graphically in Figure 12.2
and logically below:

Meet(i, j) ⇔End(i)=Begin(j)
Before(i, j) ⇔End(i) < Begin(j)
After (j, i) ⇔Before(i, j)
During(i, j) ⇔Begin(j) < Begin(i) < End(i) < End(j)
Overlap(i, j) ⇔Begin(i) < Begin(j) < End(i) < End(j)
Begins(i, j) ⇔Begin(i) = Begin(j)
Finishes(i, j) ⇔End(i) = End(j)
Equals(i, j) ⇔Begin(i) = Begin(j) ∧ End(i) = End(j)

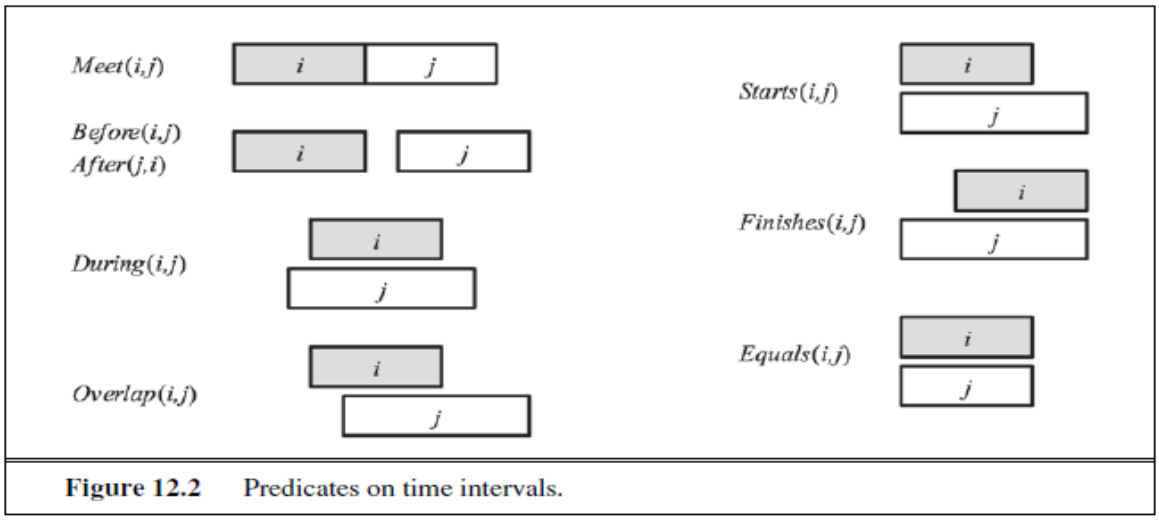


Figure 12.2 Predicates on time intervals.

Fluents and objects

Physical objects can be viewed as generalized events, in the sense that a physical object is a chunk of space–time.

The only possibility is that President(USA) denotes a single object that consists of different people at different times.

It is the object that is George Washington from 1789 to 1797, John Adams from 1797 to 1801, and so on, as in Figure 12.3.

To say that George Washington was president throughout 1790, we can write T(Equals(President(USA), GeorgeWashington),AD1790) .

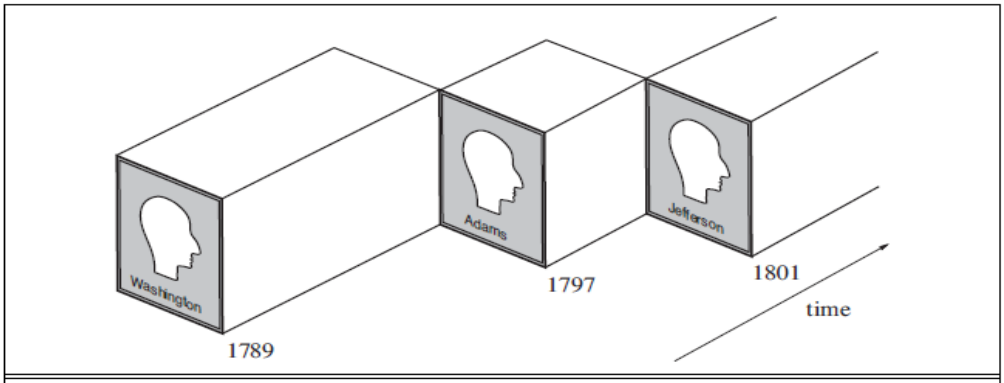


Figure 12.3 A schematic view of the object *President(USA)* for the first 15 years of its existence.

4.9 The Internet Shopping world:

In this section , we will create a shopping research agent that helps a buyer find product offers on the Internet .

The shopping agent is given a product description by the buyer and has the task of producing a list of Web pages that offer such a product for sale and ranking for the best.

Example Online Store

Select from our fine line of products:

- Computers
- Cameras
- Books
- Videos
- Music

```
<h1>Example Online Store</h1>
<i>Select</i> from our fine line of products:
<ul>
<li> <a href="http://example.com/compu">Computers</a>
<li> <a href="http://example.com/camer">Cameras</a>
<li> <a href="http://example.com/books">Books</a>
<li> <a href="http://example.com/video">Videos</a>
<li> <a href="http://example.com/music">Music</a>
</ul>
```

The above figure shows a web page and a corresponding HTML character string.

- The Agents first task is to find relevant product offers.
- Let us consider query be a product description that the user types in (e.g.,"laptops");then a page is relevant offer for query if the page is relevant and the page is indeed an offer. Also keep track of URL associated with the page.

RelevantOffer(page,url,query) ⇔ Relevant(page,url,query)^Offer(page).

- We can say a page is an offer if it contains the word “buy” or “price” within an HTML link or form on the page.
- If the page contains a string of the form “<a...buy...</a” then it is an offer,it could also be say “price” instead of “buy” or use “form” instead of “a”. We can write axioms for this:

Offer(page) ⇔ (InTag(“a”,str,page)v InTag(“form”,str,page)) ^ (In(“buy”,str) v In(“price”,str)).

- We need to find the relevant pages.
- The strategy is to start at the home page of an online store and consider all the pages that can be reached by the following relevant links.
- The Agent will have a knowledge of a number of stores, for example:

$$Amazon \in OnlineStores \wedge Homepage(Amazon, \text{“}amazon.com\text{”}) .$$
$$Ebay \in OnlineStores \wedge Homepage(Ebay, \text{“}ebay.com\text{”}) .$$
$$ExampleStore \in OnlineStores \wedge Homepage(ExampleStore, \text{“}example.com\text{”}) .$$

- These stores classify their goods into product categories, and provide links to the Major categories from their home page.
- Minor categories can be reached by following a chain of relevant links, and eventually we will reach offers .
- A page is relevant to the query if it can be reached by a chain of relevant category links from a stores home page, and following one more link to the product offer :

$$\begin{aligned} &Relevant(page, query) \Leftrightarrow \\ &\quad \exists\, store, home \;\; store \in OnlineStores \wedge Homepage(store, home) \\ &\quad \wedge \exists\, url, url_2 \;\; RelevantChain(home, url_2, query) \wedge Link(url_2, url) \\ &\qquad \wedge page = Contents(url) . \end{aligned}$$

- A chain of links between two URLs, start and end ,is relevant to a description d if the anchor text of each link is relevant category name for d.
- The existence of the chain itself is determined by a recursive definition , with the empty chain (start = end)as the base case:
- First we need to relate strings to the categories they name.
- This is done by using the predicate Name(s,c), which says that string s is a name for category c –for example , we might assert that Name(“laptops”,LaptopComputers).

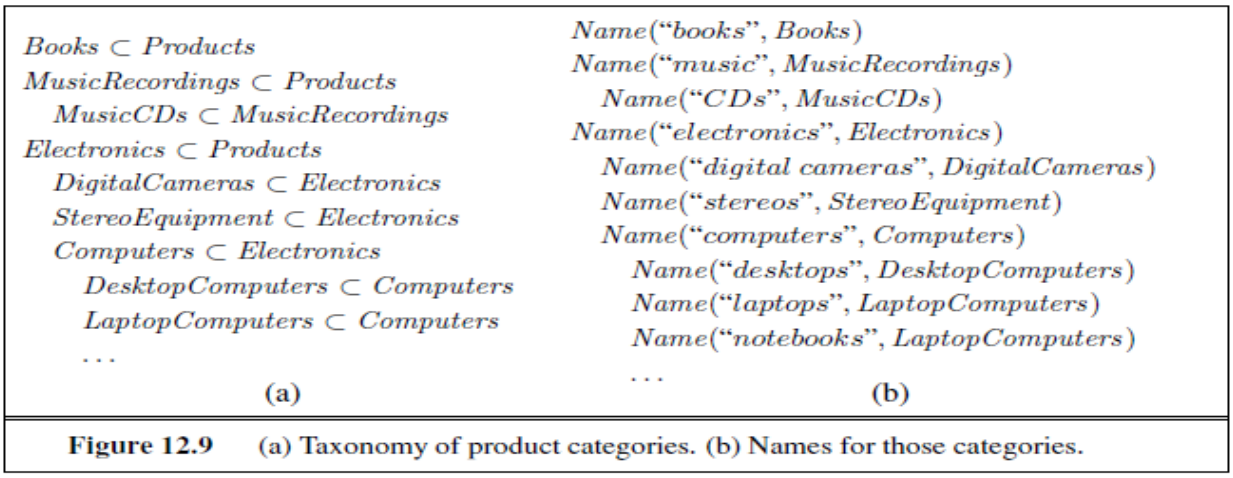


Figure 12.9 (a) Taxonomy of product categories. (b) Names for those categories.

Suppose the query is "laptops" , then RelevantCategoryName(query,text) is true when one of the following holds:

- The text and query name the same category—e.g., "laptop computers" and "laptops".
- The text names a supercategory such as "computers".
- The text names a subcategory such as "ultralight notebooks".

The logical definition of RelevantCategoryName is as follows:

RelevantCategoryName(query,text) ⇔ ∃ c1, c2 Name(query, c1) ^Name(text, c2) ∧ (c1 ⊆ c2 ∨ c2 ⊆ c1)

Comparing offers:

- To compare offers ,the agent must extract the relevant information –price ,speed ,disk size ,weight, and so on—from the offer pages.
- This can be difficult task with real web pages. A common way of dealing with this problem is to use programs called wrappers to extract information from a page.
- Consider given a page on the gen-store.com site with the text

YVM ThinkBook 970. Our price: $1449.00

Followed by various technical specifications , we would like a wrapper to extract information such as the following:

$$\exists c, offer \;\; c \in LaptopComputers \wedge offer \in ProductOffers \wedge Manufacturer(c, IBM) \wedge Model(c, ThinkBook970) \wedge ScreenSize(c, Inches(14)) \wedge ScreenType(c, ColorLCD) \wedge MemorySize(c, Gigabytes(2)) \wedge CPUSpeed(c, GHz(1.2)) \wedge OfferedProduct(offer, c) \wedge Store(offer, GenStore) \wedge URL(offer, \text{"example.com/computers/34356.html"}) \wedge Price(offer, \$(399)) \wedge Date(offer, Today) .$$

The final task is to compare the offers that have been extracted . For example , consider these three offers:

A : 1.4 GHz CPU, 2GB RAM, 250 GB disk, $299 .
B : 1.2 GHz CPU, 4GB RAM, 350 GB disk, $500 .
C : 1.2 GHz CPU, 2GB RAM, 250 GB disk, $399 .

C is dominated by A; that is , A is cheaper and faster ,and they are otherwise the same.

The shopping agent we have described here is a simple one ; many refinements are possible.

4.10 Reasoning Systems For Categories:

Categories are the primary building blocks of large-scale knowledge representation schemes.

This section describes systems specially designed for organizing and reasoning with categories.

There are two closely related families of systems:
semantic networks provide graphical aids for visualizing a knowledge base and efficient algorithms for inferring properties of an object on the basis of its category membership;

and **description logics** provide a formal language for constructing and combining category definitions and efficient algorithms for deciding subset and superset relationships between categories.

Semantic networks:

In 1909, Charles S. Peirce proposed a graphical notation of nodes and edges called existential
graphs that he called "the logic of the future."

- There are many variants of semantic networks, but all are capable of representing individual
  objects, categories of objects, and relations among objects.

- A typical graphical notation displays object or category names in ovals or boxes, and connects them with labelled links .

For example, Figure below has a MemberOf link between Mary and FemalePersons ,corresponding to the logical assertion Mary $\in$ FemalePersons ; similarly, the SisterOf link between Mary and John corresponds to the assertion SisterOf (Mary, John). We can connect categories using SubsetOf links, and so on.

It is such fun drawing bubbles and arrows that one can get carried away.

For example, we know that persons have female persons as mothers, so can we draw a HasMother link from Persons to FemalePersons?

The answer is no, because HasMother is a relation between a person and his or her mother, and categories do not have mothers.

For this reason, we have used a special notation—the double-boxed link—in Figure below.

This link asserts that

$\forall x \; x \in \text{Persons} \Rightarrow [\forall \; y \; \text{HasMother}(x, y) \Rightarrow y \in \text{FemalePersons}]$.

We might also want to assert that persons have two legs—that is,

$\forall x \; x \in \text{Persons} \Rightarrow \text{Legs}(x, 2)$.

The single-boxed link in Figure below is used to assert properties of every member of a category.

The simplicity and efficiency of this inference mechanism, compared with logical theorem proving, has been one of the main attractions of semantic networks.

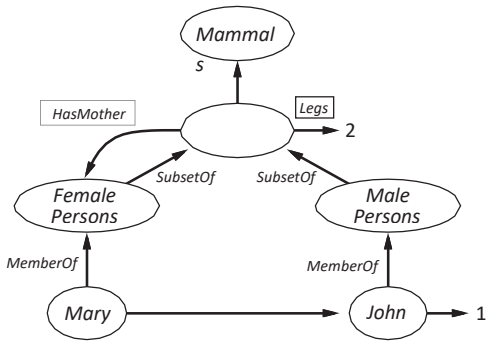


Figure 12.5 A semantic network with four objects (John, Mary, 1, and 2) and four categories. Relations are denoted by labeled links.

.

Another common form of inference is the use of inverse links.for example :HasSister is the inverse of SisterOf,which means that

The reader might have noticed an obvious drawback of semantic network notation, compared to first-order logic: the fact that links between bubbles represent only binary relations.

For example, the sentence Fly(Shankar , NewYork, NewDelhi ,Yesterday) cannot be asserted directly in a semantic network.

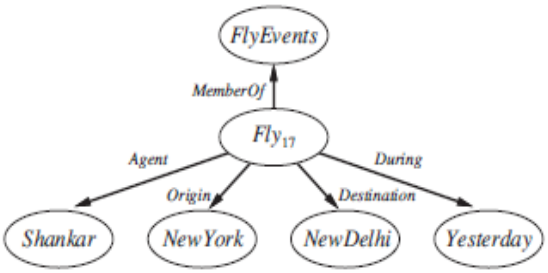


Figure 12.6 A fragment of a semantic network showing the representation of the logical assertion $Fly(Shankar, NewYork, NewDelhi, Yesterday)$.

One of the most important aspects of semantic networks is their ability to represent default values for categories.

Examining Figure 12.5 carefully, one notices that John has one leg, despite the fact that he is a person and all persons have two legs.

We say that the default is overridden by the more specific value. Notice that we could also override the default number of legs by creating a category of OneLeggedPersons, a subset of Persons of which John is a member.

We can retain a strictly logical semantics for the network if we say that the Legs assertion for Persons includes an exception for John:

$$\forall x \; x \in \text{Persons} \land x \neq \text{John} \Rightarrow \text{Legs}(x, 2) .$$

For a fixed network, this is semantically adequate.

Description logics

The syntax of first-order logic is designed to make it easy to say things about objects. Description logics are notations that are designed to make it easier to describe definitions and properties of categories.

The principal inference tasks for description logics are subsumption (checking if one category is a subset of another by comparing their definitions) and classification (checking whether an object belongs to a category)..

Some systems also include consistency of a category definition—whether the membership criteria are logically satisfiable.

```
Concept → Thing | ConceptName
        | And(Concept , . . .)
        | All(RoleName, Concept )
        | AtLeast(Integer, RoleName )
        | AtMost(Integer, RoleName )
        | Fills(RoleName , IndividualName, . . .)
        | SameAs(Path, Path)
        | OneOf(IndividualName, . . .)
   Path → [RoleName, . . .]
```

Figure 12.7 The syntax of descriptions in a subset of the CLASSIC language.

The CLASSIC language (Borgida et al., 1989) is a typical description logic. The syntax of CLASSIC descriptions is shown in Figure 12.7.

For example, to say that bachelors are unmarried adult males we would write

Bachelor = And(Unmarried, Adult ,Male) .

The equivalent in first-order logic would be

$$\text{Bachelor}(x) \Leftrightarrow \text{Unmarried}(x) \land \text{Adult}(x) \land \text{Male}(x) .$$

Any description in CLASSIC can be translated into an equivalent first-order sentence, but some descriptions are more straightforward in CLASSIC.

For example, to describe the set of men with at least three sons who are all unemployed and married to doctors, and at most two daughters who are all professors in physics or math departments, we would use

```
And(Man, AtLeast(3, Son), AtMost(2, Daughter ),
All(Son, And(Unemployed,Married, All(Spouse, Doctor ))),
All(Daughter , And(Professor , Fills(Department , Physics,Math)))) .
```

We leave it as an exercise to translate this into first-order logic.

For example, description logics usually lack negation and disjunction.

CLASSIC allows only a limited form of disjunction in the Fills and OneOf constructs.

4.11 REASONING WITH DEFAULT INFORMATION

In this section, we study defaults more generally, with a view toward understanding the semantics of defaults rather than just providing a procedural mechanism.

Circumscription and default logic:

For example, when one sees a car parked on the street, one is normally willing to believe that it has four wheels even though only three are visible.

Now, probability theory can certainly provide a conclusion that the fourth wheel exists with high probability, yet, for most people, the possibility of the car's not having four wheels does not arise unless some new evidence presents itself.

Thus, it seems that the four-wheel conclusion is reached by default, in the absence of any reason to doubt it. If new evidence arrives—for example, if one sees the owner carrying a wheel and notices that the car is jacked up—then the conclusion can be retracted.

This kind of reasoning is said to exhibit nonmonotonicity, because the set of beliefs does not grow monotonically over time as new evidence arrives.

Nonmonotonic logics have been devised with modified notions of truth and entailment in order to capture such behavior.

We will look at two such logics that have been studied extensively: circumscription and default logic.

Circumscription can be seen as a more powerful and precise version of the closedworld assumption.

The idea is to specify particular predicates that are assumed to be "as false as possible"—that is, false for every object except those for which they are known to be true.

For example, suppose we want to assert the default rule that birds fly.

We would introduce a predicate, say Abnormal 1(x), and write

$$Bird(x) \land \neg Abnormal\ 1(x) \Rightarrow Flies(x) .$$

If we say that Abnormal 1 is to be circumscribed, a circumscriptive reasoner is entitled to assume $\neg$Abnormal 1(x) unless Abnormal 1(x) is known to be true. This allows the conclusion Flies(Tweety) to be drawn from the premise Bird(Tweety).

Default logic is a formalism in which default rules can be written to generate contingent, nonmonotonic conclusions. A default rule looks like this:

$$Bird(x) : Flies(x)/Flies(x) .$$

This rule means that if Bird(x) is true, and if Flies(x) is consistent with the knowledge base, then Flies(x) may be concluded by default. In general, a default rule has the form

P : J1, . . . , Jn/C

where P is called the prerequisite, C is the conclusion, and Ji are the justifications—if any one of them can be proven false, then the conclusion cannot be drawn.

4.12 Truth maintenance systems:

- We have seen that many of the inferences drawn by a knowledge representation system will have only default status, rather than being absolutely certain.
- Inevitably, some of these inferred facts will turn out to be wrong and will have to be retracted in the face of new information. This process is called belief revision.
- Truth maintenance systems, or TMSs, are designed to handle exactly these kinds of complications.
- One simple approach to truth maintenance is to keep track of the order in which sentences are told to the knowledge base by numbering them from P1 to Pn.
- A more efficient approach is the justification-based truth maintenance system, or JTMS.

In a JTMS, each sentence in the knowledge base is annotated with a justification consisting of the set of sentences from which it was inferred.

For example, if the knowledge base already contains P ⇒ Q, then TELL(P) will cause Q to be added with the justification {P, P ⇒ Q}.

In general, a sentence can have any number of justifications. Justifications make retraction efficient. Given the call RETRACT(P), the JTMS will delete exactly those sentences for which P is a member of every justification.

The JTMS assumes that sentences that are considered once will probably be considered again, so rather than deleting a sentence from the knowledge base entirely when it loses all justifications, we merely mark the sentence as being out of the knowledge base.

If a subsequent assertion restores one of the justifications, then we mark the sentence as being back in.

In this way, the JTMS retains all the inference chains that it uses and need not rederive sentences when a justification becomes valid again.

Truth maintenance systems also provide a mechanism for generating explanations.

But explanations can also include assumptions— sentences that are not known to be true, but would suffice to prove P if they were true.

For example, one might not have enough information to prove that one's car won't start, but a reasonable explanation might include the assumption that the battery is dead.