UNIT-II

2.1 Introduction

2.1.1 Knowledge Representation and Reasoning

Intelligent agents should have capacity for:

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

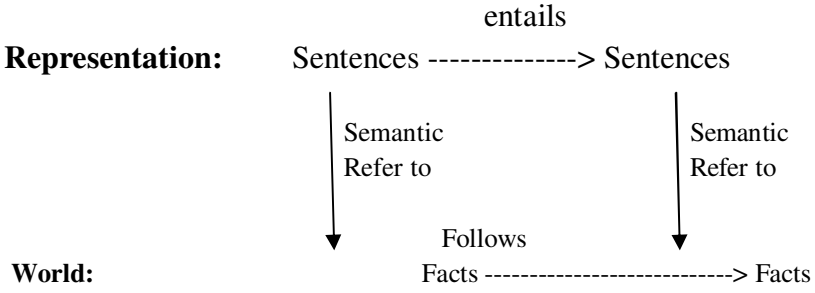
• **Knowledge Representation**, that is, representing its understanding of the world,

• **Reasoning**, that is, inferring the implications of what it knows and of the choices it has, and

• **Acting**, that is, choosing what it wants to do and carry it out.

- Representation of knowledge and the reasoning process are central to the entire field of artificial intelligence. The primary component of a knowledge-based agent is its knowledge-base. A knowledge-base is a set of sentences.
- Each sentence is expressed in a language called the knowledge representation language. Sentences represent some assertions about the world.
- There must mechanisms to derive new sentences from old ones. This process is known as inferencing or reasoning.
- Inference must obey the primary requirement that the new sentences should follow logically from the previous ones. *Logic* is the primary vehicle for representing and reasoning about knowledge.
- Specifically, we will be dealing with formal logic. The advantage of using formal logic as a language of AI is that it is precise and definite. This allows programs to be written which are declarative - they describe what is true and not how to solve problems.
- This also allows for automated reasoning techniques for general purpose inferencing. This, however, leads to some severe limitations.
- Clearly, a large portion of the reasoning carried out by humans depends on handling knowledge that is uncertain. Logic cannot represent this uncertainty well. Similarly, natural language reasoning requires inferring hidden state, namely, the intention of the speaker.
- When we say, "One of the wheel of the car is flat.", we know that it has three wheels left. Humans can cope with virtually infinite variety of utterances using a finite store of commonsense knowledge.
- Formal logic has difficulty with this kind of ambiguity.
- A logic consists of two parts, a language and a method of reasoning. The logical language, in turn, has two aspects, syntax and semantics. Thus, to specify or define a particular logic, one needs to specify three things:

- **Syntax:** The atomic symbols of the logical language, and the rules for constructing well-formed, non-atomic expressions (symbol structures) of the logic. Syntax specifies the symbols in the language and how they can be combined to form sentences. Hence facts about the world are represented as sentences in logic.
- **Semantics:** The meanings of the atomic symbols of the logic, and the rules for determining the meanings of non-atomic expressions of the logic. It specifies what facts in the world a sentence refers to. Hence, also specifies how you assign a truth value to a sentence based on its meaning in the world. A **fact** is a claim about the world, and may be true or false.
- **Syntactic Inference Method:** The rules for determining a subset of logical expressions, called theorems of the logic. It refers to mechanical method for computing (deriving) new (true) sentences from existing sentences.
- **Facts** are claims about the world that are True or False, whereas a **representation** is an expression (sentence) in some language that can be encoded in a computer program and stands for the objects and relations in the world. We need to ensure that the representation is consistent with reality, so that the following figure holds:



- There are a number of logical systems with different syntax and semantics. We list below a few of them.
    - Propositional logic
        - All objects described are fixed or unique
        - "John is a student" student(john)
        - Here John refers to one unique person.
    - First order predicate logic
        - Objects described can be unique or variables to stand for a unique object
        - "All students are poor"
        - ForAll(S) [student(S) -> poor(S)]
        - Here S can be replaced by many different unique students.
        - This makes programs much more compact:
            - eg. ForAll(A,B)[brother(A,B) -> brother (B,A)]

- replaces half the possible statements about brothers

2.2 Propositional Logic:

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

P means "It is hot"

Q means "It is humid"

R means "It is raining"

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

  (P ^ Q) => R (here meaning "If it is hot and humid, then it is raining")

  Q => P (here meaning "If it is humid, then it is hot")

  Q (here meaning "It is humid.")

- Given the truth values of all of the constituent symbols in a sentence, that sentence can be "evaluated" to determine its truth value (True or False). This is called an **interpretation** of the sentence.
- A **model** is an interpretation (i.e., an assignment of truth values to symbols) of a set of sentences such that each sentence is True. A model is just a formal mathematical structure that "stands in" for the world.
- A **valid** sentence (also called a **tautology**) is a sentence that is True under *all* interpretations. Hence, no matter what the world is actually like or what the semantics is, the sentence is True. For example "It's raining or it's not raining."
- An **inconsistent** sentence (also called **unsatisfiable** or a **contradiction**) is a sentence that is False under *all* interpretations. Hence the world is never like what it describes. For example, "It's raining and it's not raining."
- Sentence P **entails** sentence Q, written P |= Q, means that whenever P is True, so is Q. In other words, all models of P are also models of Q.

2.3 The Language:

I will describe the elements of the propositional calculus formally. It's best for the moment not to attempt to associate meanings with elements of this language; think of what we are doing now as describing the rules of a meaningless game. Later, we'll talk about meaning. Here are the elements of the language:[1]

Atoms: the two distinguished atoms `T` and `F` and the countably infinite set of those strings of characters that begin with a capital letter, for example, `P`, `Q`, `R`, . . . , `P1`, `P2`, `ON_A_B`, and so on.

Connectives: $\vee$, $\wedge$, $\supset$, and $\neg$, called "or," "and," "implies," and "not," respectively. (Later, I give these connectives meanings related to their names; for now, they are meaningless symbols.)

Syntax of ***well-formed formulas (wffs)***, also called ***sentences***:

- Any atom is a wff.

  Examples: `P`, `R`, `P3`

- If ω_1 and ω_2 are wffs, so are

  $\omega_1 \vee \omega_2$ (called a ***disjunction*** of ω_1 and ω_2)

  $\omega_1 \wedge \omega_2$ (called a ***conjunction*** of ω_1 and ω_2)

  $\omega_1 \supset \omega_2$ (called an ***implication***)

  $\neg\omega_1$ (called a *negation* of ω_1)

Atoms and atoms with a $\neg$ sign in front of them are called *literals*. In $\omega_1 \supset \omega_2$, ω_1 is called the *antecedent* of the implication, and ω_2 is called the *consequent* of the implication.

There are no other wffs.

Example: `P` $\supset \neg\neg$ is not a wff.

Examples of wffs:

$(P \wedge Q) \supset \neg P$

$P \supset \neg P$

$P \vee P \supset P$

$(P \supset Q) \supset (\neg Q \supset \neg P)$

$\neg\neg P$

Note the use of extra-linguistic separators, (and), in the preceding examples. They group wffs into (sub) wffs according to the recursive definitions. Some treatments of logic formalize separators as part of the language, but I will use them somewhat loosely and intuitively. Wffs can be recognized by using their definitions recursively. For example, I will show that $(P \wedge Q) \supset \neg R$ is a wff. First, note that since P and Q are wffs, so is $(P \wedge Q)$. And $\neg R$ is a wff because R is. Therefore, $(P \wedge Q) \supset \neg R$ is a wff.

2.4 rules of inference

We have a number of ways by which we can produce additional wffs from other ones. These are called *rules of inference*. A rule of inference typically has the form: γ can be *inferred* from α (or from α and β).[2] Here are some commonly used rules of inference:

- The wff ω_2 can be inferred from the wffs ω_1 and $\omega_1 \supset \omega_2$ (this rule of inference is called *modus ponens*).
- The wff $\omega_1 \wedge \omega_2$ can be inferred from the two wffs ω_1 and ω_2 ($\wedge$ introduction).
- The wff $\omega_2 \wedge \omega_1$ can be inferred from the wff $\omega_1 \wedge \omega_2$ (commutativity of $\wedge$).
- The wff ω_1 can be inferred from the wff $\omega_1 \wedge \omega_2$ ($\wedge$ elimination).
- The wff $\omega_1 \vee \omega_2$ can be inferred either from the single wff ω_1 or from the single wff ω_2 ($\vee$ introduction).
- The wff ω_1 can be inferred from the wff $\neg(\neg\omega_1)$ ($\neg$ elimination).

2.5 definition of proof

The sequence of wffs $\{\omega_1, \omega_2, \ldots, \omega_n\}$ is called a *proof* (or a *deduction*) of ω_n from a set of wffs Δ iff each ω_i in the sequence is either in Δ or can be inferred from a wff (or wffs) earlier in the sequence by using one of the rules of inference. If there is a proof of ω_n from Δ, we say that ω_n is a *theorem* of the set Δ. I use the following notation for expressing that ω_n can be proved from Δ:

$\Delta \vdash \omega_n$

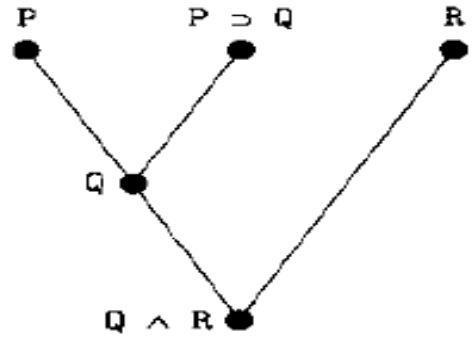
The concept of proof and theoremhood is relative to a particular set of inference rules used. If we denote the set of inference rules by the letter $\mathcal{R}$, we sometimes then write the fact that ω_n can be proved from Δ using the inference rules in $\mathcal{R}$ by

$\Delta \vdash_{\mathcal{R}} \omega_n$

Example: Given a set, Δ, of wffs: $\{P, R, P \supset Q\}$, the following sequence is a proof of $Q \wedge R$ given the inference rules just presented:

$\{P, P \supset Q, Q, R, Q \wedge R\}$.

The concept of proof can be based on a partial order as well as on a sequence. This partial order can be represented by a tree structure. Each node in the proof tree is labeled by a wff and must correspond either to a wff in Δ or be inferable from its parents in the tree using one of the rules of inference. The labeled tree is a proof of the label of the root node. Below figure shows a sample proof tree:



2.5 Semantics:

2.5.1 Interpretations:

Now it's time to talk about "meanings." *Semantics* has to do with associating elements of a logical language with elements of a domain of discourse. Such associations are what we mean by "meaning." For propositional logic, we associate atoms with *propositions* about the world. (Thus the name *propositional* logic.) So, for example, we might associate the atom `BAT_OK` with the proposition "The battery is charged." (We are not *compelled* to make the association suggested by a mnemonic atom string; we could make other ones instead.) An association of atoms with propositions is called an *interpretation*. In a given interpretation, the proposition associated with an atom is called the *denotation* of that atom.

Under a given interpretation, atoms have *values*—*True* or *False*. If atom α is associated with proposition P, then we say that α has value *True* just in case P is true of the world; otherwise, it has value *False*. The special atom `T` always has value *True*, and the special atom `F` always has value *False*. Since propositions about the world must either be true or false (an idealization we are willing to make here), we can specify an interpretation by assigning values directly to the atoms in a language—regardless of which proposition about the world each atom denotes.

If an agent's sensory apparatus for determining the truth or falsity of various propositions about the world is reliable, then when sensed feature x_1 has value 1, the corresponding proposition about the world will be true, and the associated propositional logic atom, perhaps X1, will have value *True*. For this reason, instead of representing sensed information for an agent by a value of 1 or 0 on a certain input "wire," we can represent it by a propositional calculus atom in an agent's memory structure that we call its *knowledge base*. The explicit occurrence of an atom, say, X1, in the knowledge base of an agent is intended to mean that the agent regards an associated proposition to be true in its world. We will soon see how an agent uses wffs in its knowledge base.

ω_1	ω_2	$\omega_1 \wedge \omega_2$	$\omega_1 \vee \omega_2$	$\neg\omega_1$	$\omega_1 \supset \omega_2$
True	*True*	*True*	*True*	*False*	*True*
True	*False*	*False*	*True*	*False*	*False*
False	*True*	*False*	*True*	*True*	*True*
False	*False*	*False*	*False*	*True*	*True*

The Propositional Truth Table

2.5.2 The Propositional Truth Table:

Given the values of atoms under some interpretation, we can use a *truth table* to compute a value for any wff under that same interpretation. The truth table establishes the semantics (meanings) of the propositional connectives. Let ω_1 and ω_2 be wffs; then the truth table rules are

- $(\omega_1 \wedge \omega_2)$ has value *True* if both ω_1 and ω_2 have value *True*; otherwise, it has value *False*.
- $(\omega_1 \vee \omega_2)$ has value *True* if one or both of ω_1 or ω_2 has value *True*; otherwise, it has value *False*.
- $\neg\omega_1$ has value *True* if ω_1 has value *False*; otherwise, it has value *False*.
- The semantics of $\supset$ is defined in terms of $\vee$ and $\neg$. Specifically, $(\omega_1 \supset \omega_2)$ is an alternative and equivalent form of $(\neg\omega_1 \vee \omega_2)$.

These are called truth table rules because they are usually presented in tabular form, as they are shown in above table format.

We can use the truth table to compute the value of any wff given the values of the constituent atoms in the wff. As an example of the use of the truth table to compute the value of a wff, suppose P has value *False*, Q has value *False*, and R has value *True*. In this interpretation, what is the truth value of [(P ⊃ Q) ⊃ R] ⊃ P? Working from the "inside out," we first compute the value of P ⊃ Q to be *True*; then we see that (P ⊃ Q) ⊃ R is *True*. Finally, since P is *False*, the value of the entire expression must be *False*.

If an agent describes its world using n features (corresponding to propositions), and these features are represented in the agent's model of the world by a corresponding set of n atoms, then there are 2^n different ways its world can be—so far as the agent can discern—because there are 2^n ways in which the n atoms can have values of *True* and *False*. Each of the ways the world can be corresponds to an interpretation. Given values for the n atoms (an interpretation), the agent can use the truth table to find the values of any wffs. What about the reverse process? Suppose we are given the values of the wffs in a set of wffs. Do those values induce a unique interpretation? This question is important because often an agent is given a set of constraints among features (expressed as wffs that have value *True*). Can the agent then induce an assignment of values (an interpretation) to the atoms in its language and thus determine whether or not the corresponding propositions about the world are true or false? That is, do the formulas specify one of the 2^n ways the world can be? The answer is usually no. Instead, there

May be many interpretations that give each wff in a set of wffs the value true.

2.5.2 Satisfiabilty and models:

We say that an interpretation *satisfies* a wff if the wff is assigned the value *True* under that interpretation. An interpretation that satisfies a wff is called a *model* of that wff. An interpretation that satisfies each wff in a set of wffs is called a *model* of that set of wffs. Trivially, we can tell whether or not an interpretation satisfies any atom because an interpretation assigns a value to each atom. We can use the truth table to tell whether or not an interpretation satisfies a wff that comprises atoms.

In my block-lifting robot, I expressed a constraint among some features by the following wff:

BAT_OK ∧ LIFTABLE ⊃ MOVES

If this wff is to have value *True*, as we intend, then we must rule out all interpretations in which BAT_OK and LIFTABLE have value *True* and MOVES has value *False*. Each "constraint wff" tells us something about the way the world can be and thus rules out some of the possible models. (Each model corresponds to one of the ways the world can be.) In general, the more wffs we have that describe the world, the fewer models! This fact shouldn't be surprising; the more we know about the world, the less uncertainty there is about the way it could be.

It is possible that *no* interpretation satisfies a wff (or a set of wffs). In that case, the wff (or the set of wffs) is said to be *inconsistent* or *unsatisfiable*. Examples of unsatisfiable wffs are F and P ∧ ¬P (no interpretation makes these wffs *True*). An example of an unsatisfiable set of wffs is {P ∨ Q, P ∨ ¬Q, ¬P ∨ Q, ¬P ∨ ¬Q}. (Use the truth table to confirm that no interpretation makes *all* of these wffs *True*.)

2.5.3 Validity:

A wff is said to be *valid* if it has value *True* under *all* interpretations of its constituent atoms. (Thus, a valid wff is vacuous; it tells us nothing about the way the world can be.) Here are some examples of valid wffs.

- P ⊃ P (which is the same as ¬P ∨ P)
- T
- ¬(P ∧ ¬P)
- Q ∨ T
- [(P ⊃ Q) ⊃ P] ⊃ P
- P ⊃ (Q ⊃ P)

Use of the truth table to determine the validity of a wff takes time exponential in the number of atoms because the wff must be evaluated for all combinations of the values of the atoms.

2.5.4 Equivalence:

Two wffs are said to be *equivalent* if and only if their truth values are identical under *all* interpretations. I write equivalence with a ≡ sign. We can use the truth table to prove the following equivalences:

- DeMorgan's laws:

$$\neg(\omega_1 \vee \omega_2) \equiv \neg\omega_1 \wedge \neg\omega_2$$

$$\neg(\omega_1 \wedge \omega_2) \equiv \neg\omega_1 \vee \neg\omega_2$$

- Law of the contrapositive:

$$(\omega_1 \supset \omega_2) \equiv (\neg\omega_2 \supset \neg\omega_1)$$

- If ω_1 and ω_2 are equivalent, then the following formula is valid:

$$(\omega_1 \supset \omega_2) \wedge (\omega_2 \supset \omega_1)$$

Because of this fact, the notation $\omega_1 \equiv \omega_2$ is often used as an abbreviation for $(\omega_1 \supset \omega_2) \wedge (\omega_2 \supset \omega_1)$.

2.5.5 Entailment:

If a wff ω has value *True* under all of those interpretations for which each of the wffs in a set Δ has value *True*, then we say that Δ *logically entails* ω and that ω *logically follows* from Δ and that ω is a *logical consequence* of Δ. I use the symbol $\models$ to denote logical entailment and write $\Delta \models \omega$. Here are some examples.

- $\{P\} \models P$
- $\{P, P \supset Q\} \models Q$
- $F \models \omega$ (where ω is *any* wff)
- $P \wedge Q \models P$

Logical entailment is important in AI because it provides a very strong way of showing that if certain propositions are true about a world then some other proposition of interest (perhaps one that cannot be sensed) must also be true. For example, suppose we sense features that we associate with the formulas BAT_OK, and ¬MOVES, and that we represent some of our knowledge about the world by the formula BAT_OK ∧ LIFTABLE ⊃ MOVES. That is, we have three formulas, two of which describe a particular world situation and one of which describes general knowledge about the world. I leave it to you to establish by the truth table that ¬LIFTABLE is logically entailed by these three formulas. Since, by this entailment, ¬LIFTABLE has value *True* in *all* of the interpretations in which the three formulas have value *True*, it must *a fortiori* have value *True* in our *intended interpretation* (that is, the one we are associating with the robot's world). Therefore, the proposition that is part of our intended interpretation, namely, that "the block is not liftable," must be true!

Because entailment is such a powerful tool for determining the truth or falsity of propositions about the world, it is important for us to study how to represent information as wffs and how to produce entailed wffs efficiently. We can always do this using the truth table method, but we seek simpler methods. An attractive substitute for entailment is inference. Although they are fundamentally different concepts, they are linked by the concepts of soundness and completeness.

2.5.6 Soundness and Completeness:

There are two important definitions that inference with entailment:

1. If, for any set of wffs, Δ, and wff, ω, $\Delta \vdash_{\mathcal{R}} \omega$ implies $\Delta \models \omega$, we say that the set of inference rules, $\mathcal{R}$, is *sound*.
2. If, for any set of wffs, Δ, and wff, ω, it is the case that whenever $\Delta \models \omega$, there exists a proof of ω from Δ using the set of inference rules, $\mathcal{R}$, we say that $\mathcal{R}$ is *complete*.

When inference rules are sound and complete, we can determine whether one wff follows from a set of wffs by searching for a proof instead of by using the truth table. When the inference rules are sound, if we find a proof of ω from Δ, we know that ω logically follows from Δ. When the inference rules are complete, we know that we will eventually be able to confirm that ω follows from Δ (when it does so) by using a complete search procedure to search for a proof. Substituting proof methods for truth table methods usually gives great computational advantage in both the propositional calculus and the predicate calculus (which we will study later).

To determine whether or not a wff logically follows from a set of wffs or can be proved from a set of wffs is, in general, an NP-hard problem [Cook 1971]. (That is, its complexity cannot be proven to be less than exponential in the number of atoms.) Nevertheless, there are special cases that are tractable, and so it is important to understand the process of logical reasoning.

2.6 Resolution in the Propositional Calculus:

2.6.1 A New Rule of Inference: Resolution

Clauses as WFFs:

- Many of the inference rules can be combined into one rule, called resolution.
- The version of resolution that applies to a special format for WFFs called clauses.
- To understand this clause:-
    - First, recall that a literal is either an atom or the negation of an atom.
    - A clause is a set of literals.
    - The set is an abbreviation for the disjunction of all the literals in the set. Thus clause is a special kind of WFF.
    - Example: the clause {P,Q,¬R} (equivalent to {P∨Q∨¬R} is a WFF.
    - The empty clause {} sometimes written as nil is equivalent to 'F'. (whose value is false).

2.6.2 Resolution on Clauses:

The resolution rule for the propositional calculus can be stated as follows: from $\{\lambda\} \cup \Sigma_1$ and $\{\neg\lambda\} \cup \Sigma_2$ (where Σ_1 and Σ_2 are sets of literals and λ is an atom), we can infer $\Sigma_1 \cup \Sigma_2$, which is called the *resolvent* of the two clauses. The atom λ is the *atom resolved upon*, and the process is called *resolution*.

Here are some examples.

- Resolving $R \vee P$ and $\neg P \vee Q$ yields $R \vee Q$. The two clauses being resolved can be rewritten as the implications $\neg R \supset P$ and $P \supset Q$. A rule of inference called *chaining* applied to these implications yields $\neg R \supset Q$, which is equivalent to the resolvent $R \vee Q$. Thus we see that chaining is a special case of resolution.

- Resolving R and $\neg R \vee P$ yields P. Since the second clause is equivalent to $R \supset P$, we see that *modus ponens* is also a special case of resolution.
- Resolving $P \vee Q \vee R \vee S$ with $\neg P \vee Q \vee W$ on P yields $Q \vee R \vee S \vee W$. Note that only one instance of Q appears in the resolvent—which was, after all, defined to be a set!
- Resolving $P \vee Q \vee \neg R$ with $P \vee W \vee \neg Q \vee R$ on Q yields $P \vee \neg R \vee R \vee W$. Resolving them on R yields $P \vee Q \vee \neg Q \vee W$. In this case, since both $\neg R \vee R$ and $Q \vee \neg Q$ have value *True*, the value of each of these resolvents is *True*. In this example, we must resolve either on Q or on R—not on both simultaneously! That is, $P \vee W$ is not a resolvent of the two clauses!

Resolving λ, a positive literal, with $\neg\lambda$ produces the empty clause. From λ and $\neg\lambda$ we can infer F because λ and $\neg\lambda$ are contradictory. Any set of wffs containing λ and $\neg\lambda$ is unsatisfiable. On the other hand, a clause that contains an atom and its negation (such as $\lambda \vee \neg\lambda$) has value *True* regardless of the value of λ.

2.6.4 Soundness of Resolution:

The resolution rule I have just presented is a sound rule of inference. That is, if the clauses $\{\lambda\} \cup \Sigma_1$ and $\{\neg\lambda\} \cup \Sigma_2$ both have value *True*, then their resolvent, namely, $\Sigma_1 \cup \Sigma_2$, does also. One way to prove that fact is to do "reasoning by cases." We know that the atom λ has either value *True* or value *False*. If (case 1) λ has value *True*, then $\neg\lambda$ has value *False*, and the clause Σ_2 must have value *True* in order for the clause $\{\neg\lambda\} \cup \Sigma_2$ to be *True*. If (case 2) λ has value *False*, then the clause Σ_1 must have value *True* in order for the clause $\{\lambda\} \cup \Sigma_1$ to have value *True*. Combining these two cases, we see that either Σ_1 or Σ_2 must have value *True*; hence $\Sigma_1 \cup \Sigma_2$ has value *True*. A similar argument can also be carried out using the truth table.

2.7 Converting Arbitrary WFFs to Conjunctions of Clauses:

Any wff in the propositional calculus can be converted to an equivalent conjunction of clauses. A wff written as a conjunction of clauses is said to be in *conjunctive normal form (CNF)*. (A wff written as a disjunction of conjunctions of literals is said to be in *disjunctive normal form (DNF)*.) I will use an example to illustrate the steps of the process of converting an arbitrary wff to CNF. The wff I use to illustrate this process is $\neg(P \supset Q) \vee (R \supset P)$.

1. Eliminate implication signs by using the equivalent form using $\vee$:

$\neg(\neg P \vee Q) \vee (\neg R \vee P)$

2. Reduce the scopes of ¬ signs by using DeMorgan's laws and by eliminating double ¬ signs:

$(P \wedge \neg Q) \vee (\neg R \vee P)$

3. Convert to CNF by using the associative and distributive laws. First,

$(P \vee \neg R \vee P) \wedge (\neg Q \vee \neg R \vee P)$ then $(P \vee \neg R) \wedge (\neg Q \vee \neg R \vee P)$

A conjunction of clauses (that is, the CNF form of a wff) is usually expressed as a set of clauses (with conjunction of the clauses implied); thus,

$\{(P \vee \neg R), (\neg Q \vee \neg R \vee P)\}$

The following procedure might be useful in step 3 to convert wffs (or parts of wffs) from DNF form to CNF form. First, write the DNF wffs as a matrix whose row elements are the literals in each conjunct; we have implicit disjunction of the rows. For example, the DNF form $(P \wedge Q \wedge \neg R) \vee (S \wedge R \wedge \neg P) \vee (Q \wedge S \wedge P)$ can be written in matrix form as follows:

$$\begin{matrix} P & Q & \neg R \\ S & R & \neg P \\ Q & S & P \end{matrix}$$

Now select a literal in each row and make a disjunction of these literals. In my example, one such selection might be $P \vee R \vee P$. Make all possible such selections. Each selection corresponds to a clause, and we take the conjunction of all these clauses as the CNF form of the original wff. We can simplify some of these clauses; for example, $P \vee R \vee P$ simplifies to $P \vee R$. We can eliminate some of the clauses; for example, $P \vee \neg P \vee Q$ can be eliminated because it trivially has value *True*. We can also eliminate clauses that are *subsumed* by one of the other clauses. (A clause, γ_1, *subsumes* a clause γ_2, if the literals in γ_1 are a subset of those in γ_2.) For example, $P \vee R$ subsumes both $P \vee R \vee Q$ and $P \vee R \vee S$.

2.8 Resolution Refutation:

Resolution is a sound rule of inference.

That is, $\Delta \vdash_{res} \gamma$
implies $\Delta \models \gamma$, where γ is a clause. But resolution is not complete. For example, $P \wedge R \models P \vee R$, but we cannot infer $P \vee R$ using resolution on the set of clauses $\{P, R\}$ (because there is nothing that can be resolved). Therefore, we cannot use resolution directly to decide *all* logical entailments. However, we can show by resolution that the negation of $P \vee R$ is inconsistent with the set $\{P, R\}$, and thus, using proof by contradiction, establish that $P \wedge R \models P \vee R$.

To illustrate the process, the negation of $P \vee R$ is $\neg P \wedge \neg R$. Expressed as a (conjunctive) set of clauses, the negation of what we are trying to prove is $\{\neg P, \neg R\}$. To show the inconsistency of these clauses with $P \wedge R$, we add P and R to that set to obtain $\{\neg P, \neg R, P, R\}$. Resolving on members of this latter set produces the empty clause, which is a contradiction, and thus we have indirectly established $P \vee R$ from $P \wedge R$.

In general, a resolution refutation for proving an arbitrary wff, ω, from a set of wffs, Δ, proceeds as follows:

1. Convert the wffs in Δ to clause form—a (conjunctive) set of clauses.
2. Convert the negation of the wff to be proved, ω, to clause form.
3. Combine the clauses resulting from steps 1 and 2 into a single set, Γ.
4. Iteratively apply resolution to the clauses in Γ and add the results to Γ either until there are no more resolvents that can be added or until the empty clause is produced.

- Completeness of resolution refutation: the empty clause will be produced by the resolution refutation procedure if $\Delta \models \omega$. Thus, we say that propositional resolution is *refutation complete*.
- Decidability of propositional calculus by resolution refutation: if Δ is a finite set of clauses and if $\Delta \not\models \omega$, then the resolution refutation procedure will terminate without producing the empty clause.

The reasoning we used in the block-lifting example of the last chapter can be performed using resolution refutation. We are given the set Δ of wffs:

1. BAT_OK

2. ¬MOVES

3. BAT_OK ∧ LIFTABLE ⊃ MOVES

The clause form of the third wff is

4. ¬BAT_OK ∨ ¬LIFTABLE ∨ MOVES

The negation of the wff to be proved yields another clause:

5. LIFTABLE

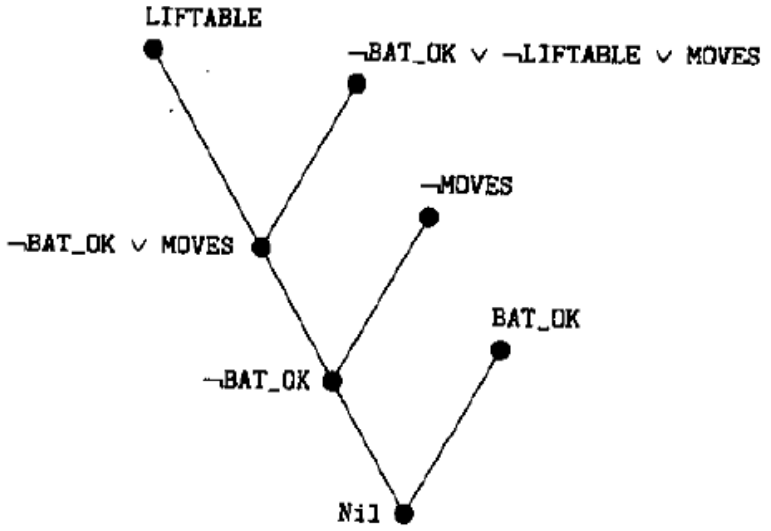
Now we perform resolutions to produce the following sequence of clauses:

6. ¬BAT_OK ∨ MOVES (from resolving 5 with 4)

7. ¬BAT_OK (from 6, 2)

8. Nil (from 6, 1)

We can also represent this refutation as a refutation tree; as shown in below figure:



2.9 Horn Clauses:

There is a special type of clause that is important in AI and in other parts of computer science. A *Horn clause* is a clause that has at most one positive literal.

Here are some examples of Horn clauses.

P, ¬P ∨ Q, ¬P ∨ ¬Q ∨ R, ¬P ∨ ¬R

Clauses of this type were first investigated by the logician Alfred Horn [Horn 1951].

There are three types of Horn clauses.

- A single atom—often called a "fact."
- An implication—often called a "rule"—whose antecedent consists of a conjunction of positive literals and whose consequent consists of a single positive literal.
- A set of negative literals—written in implication form with an antecedent consisting of a conjunction of positive literals and an empty consequent. This form is obtained, for example, when one negates a wff to be proved consisting of a conjunction of positive literals. Such a clause is therefore often called a "goal."

Examples of these three types of Horn clauses are, respectively, P, $P \wedge Q \supset R$, and $P \wedge Q \supset$.

2.10 THE PREDICATE CALCULUS:

The predicate calculus has symbols called object constants, relation constants and function constants. These language entities will be used to refer to objects in the world and to propositions about the world.

The Language and Its Syntax:

There are three different categories they are:

- Components
- Terms
- WFF's

Components:

- We have an infinite set of *object constants*. These will be strings of alphanumeric characters (often mnemonic, but that helps only *us*, not the computer). My convention will be that object constants begin with either a capital letter or a numeral.

  Examples: `Aa`, `125`, `13B`, `Q`, `John`, `EiffelTower`

- And we will have an infinite set of *function constants* of all "arities."[1] These will be strings of alphanumeric characters beginning always with a lowercase letter and (for now) superscripted by their arity.

  Examples: $\texttt{fatherOf}^1$, $\texttt{distanceBetween}^2$, $\texttt{times}^2$

- We also have an infinite set of *relation constants* of all arities. These will be strings of alphanumeric characters beginning with a capital letter and (for now) superscripted by their arity. (I sometimes call a relation constant a *predicate*.)

  Examples: $\texttt{B17}^3$, $\texttt{Parent}^2$, $\texttt{Large}^1$, $\texttt{Clear}^1$, $\texttt{X11}^4$

- I also will use the propositional connectives $\vee$, $\wedge$, $\neg$, and $\supset$, and delimiters (,), [,], and separator ,.

Terms:

- An object constant is a *term*.
- A function constant of arity *n*, followed by *n* terms in parentheses and separated by commas, is a term. This type of term is called a *functional expression*. In displaying such a term, I usually omit the arity superscript when its value is obvious from context. We might think of object constants as function constants of arity 0.

  Examples: `fatherOf(John, Bill)`, `times(4, plus(3, 6))`, `Sam`

WFF's:

- Atoms: a relation constant of arity n followed by n terms in parentheses and separated by commas is an *atom* (also called an *atomic formula*). A relation constant of arity 0 omits the parentheses. Again, I usually omit the arity superscript when its value is obvious from context.

  An atom is a wff.

  Examples: `Greaterthan(7, 2)`, `P(A, B, C, D)`, `Q`

- Propositional wffs: any expression formed out of predicate-calculus wffs in the same way that the propositional calculus forms wffs out of other wffs is a wff, called a *propositional wff*.

  Example: [`Greaterthan(7, 2)` ∧ `Lessthan(15, 4)`] ∨ ¬`Brother(John, Sam)` ∨ `P`

2.11 Semantics:

Worlds:

We now have a language that can be used to refer to objects in the world as well as to propositions. Here is what we can talk about now.

- The world can have an infinite number of *objects*, also called *individuals*, in it. These can be individuals that are rather concrete, like Block *A*, Mt. Whitney, Julius Caesar, and so on. Or they can be abstract entities like the number 7, π, the *set* of all integers, and so on. They can even be fictional or invented entities (about whose actual existence people might argue), like beauty, Santa Claus, a unicorn, honesty, and so on. As long as we are willing to give it a name and say something about it, we can think of it as an actual individual in a world we want to talk about.
- Functions on individuals. We can have an infinite number of functions of all arities that map n tuples of individuals into individuals. For example, there might be a function that maps a person into his or her father or one that maps the numbers 10 and 2 into the quotient 5.

- Relations over individuals. The individuals can participate in an arbitrary number of relations. These will each have arities. (A relation of arity 1 is called a *property*.) Thus, individuals might have properties such as heavy, big, blue, and so on, and they might participate in relations like bigger, in between, and so on. To specify an *n*-ary relation *extensionally*, we explicitly list *all* of the *n* tuples of individuals that participate in that relation.

Interpretations:

An *interpretation* of an expression in the predicate calculus is an assignment that maps object constants into objects in the world, *n*-ary function constants into *n*-ary functions, and *n*-ary relation constants into *n*-ary relations. These assignments are called the *denotations* of their corresponding predicate-calculus expressions. The set of objects to which object constant assignments are made is called the *domain* of the interpretation.

Given an interpretation for its component parts, an atom has the value *True* just in case the denoted relation holds for those individuals denoted by its terms. If the relation doesn't hold, the atom has value *False*.

The values *True* and *False* of nonatomic wffs are determined by the same truth table used in the propositional calculus.

Example: Consider the blocks world again. It is a world in which there exist entities, **A**, **B**, **C**, and **Floor**.[2] We will eventually be creating an assignment that maps elements of the predicate calculus into these world objects. Since we cannot actually make the world objects themselves be the values of assignments, we imagine that the world is a mathematical structure, which (among other things) contains the mathematical objects **A**, **B**, **C**, and **Floor**. (Try not to be bothered by the statement "the world is a mathematical structure." Whatever the world *really* is, it won't matter for our purposes that we consider it to be a mathematical structure.)

We also imagine relations among these objects. For example, we might have the relations **On** and **Clear**. These relations can be defined extensionally by the *n* tuples of objects that participate in them. For example, suppose we have the configuration of blocks shown in Figure 15.1. In this world, the relation **On** is given by the pairs <**B**, **A**>, <**A**, **C**>, and <**C**, **Floor**>. The relation **Clear** is given by the singleton, <**B**>.

So, the individuals **A**, **B**, **C**, and **Floor**, and the relations **On** and **Clear** constitute the blocks world in this example. To describe that world in the predicate calculus, I employ the object constants `A`, `B`, `C`, and `Fl`, the binary relation constant `On`, and the unary relation constant `Clear`. I am using mnemonic symbols here for convenience; probably it would be better pedagogy to use the relation constants

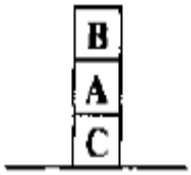


Figure 15.1 A Configuration of Blocks

Predicate Calculus	***World***
`A`	**A**
`B`	**B**
`C`	**C**
`Fl`	**Floor**
`On`	**On** = {< **B**, **A** >, < **A**, **C** >, < **C**, **Floor** >}
`Clear`	**Clear** = {< **B** >}

Table 15.1 A Mapping between Predicate Calculus and the World

G0045, G123, . . . , and so on in order to emphasize the notion that the symbols used in our predicate-calculus language do not have any preassigned meanings.

Our intended interpretation (which is just one of many possible interpretations) for these predicate-calculus expressions is given in Table 15.1. We can use these assignments to determine the value of some predicate-calculus wffs:

`On(A,B)` is *False* because <**A**,**B**> is not in the relation **On**.

`Clear(B)` is *True* because <**B**> is in the relation **Clear**.

`On(C,F1)` is *True* because <**C**, **Floor**> is in the relation **On**.

`On(C,F1)` ∧ ¬`On(A,B)` is *True* because both `On(C,F1)` and ¬`On(A,B)` are *True*.

Models and Related Notions:

Several semantic notions have the same definitions in the predicate calculus as they do in the propositional calculus. To review,

- An interpretation *satisfies* a wff if the wff has the value *True* under that interpretation.
- An interpretation that satisfies a wff is a *model* of that wff.
- Any wff that has the value *True* under *all* interpretations is *valid*.

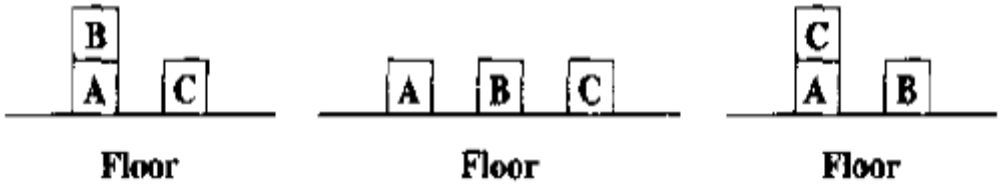


Figure 15.2 Three Blocks-World Situations

- Any wff that does not have a model is *inconsistent* or *unsatisfiable*.
- If a wff ω has value *True* under all of those interpretations for which each of the wffs in a set Δ has value *True*, then Δ *logically entails* ω ($\Delta \models \omega$).
- Two wffs are *equivalent* if and only if their truth values are identical under *all* interpretations (that is, if and only if the two wffs logically entail one another).

Suppose we wanted to say that *every* object in the domain had a certain property (or participated in a certain relation). We could do this for finite domains by a conjunction such as `Clear(B1)` ∧ `Clear(B2)` ∧ `Clear(B3)` ∧ `Clear(B4)`. But long conjunctions are cumbersome, and we can't even write down infinite ones, which might be needed if the domain were infinite.

Or suppose we wanted to say that at least one object in the domain had a certain property. For finite domains, such a statement could be made by a disjunction such as `Clear(B1)` ∨ `Clear(B2)` ∨ `Clear(B3)` ∨ `Clear(B4)`. Again, there is a problem with large or infinite domains.

2.12 Quantification:

Suppose we wanted to say that *every* object in the domain had a certain property (or participated in a certain relation). We could do this for finite domains by a conjunction such as `Clear(B1)` ∧ `Clear(B2)` ∧ `Clear(B3)` ∧ `Clear(B4)`. But long conjunctions are cumbersome, and we can't even write down infinite ones, which might be needed if the domain were infinite.

Or suppose we wanted to say that at least one object in the domain had a certain property. For finite domains, such a statement could be made by a disjunction such as `Clear(B1)` ∨ `Clear(B2)` ∨ `Clear(B3)` ∨ `Clear(B4)`. Again, there is a problem with large or infinite domains.

1. We have an infinite set of *variable symbols* consisting of strings beginning with lowercase letters near the end of the alphabet, such as `p`, `q`, `r`, `s`, `t`, . . . , `p1`, `p2`, . . . , and so on. (These will be distinguished from function constants by their use in context.) A variable symbol is a term (in addition to the terms I defined previously). Thus `f(x, Bob, C17)` is a ternary functional expression, for example.
2. We have the quantifier symbols, ∀ and ∃. ∀ is called a *universal quantifier*, and ∃ is called an *existential quantifier*.

3. If ω is a wff and ξ is a variable symbol, then both $(\forall\xi)\omega$ and $(\exists\xi)\omega$ are wffs. ξ is called the variable *quantified over*, and ω is said to be *within the scope* of the quantifier. Wffs of the form $(Q\xi)\omega$, where ξ is a variable symbol and Q is either $\forall$ or $\exists$ will most commonly (in our applications) be such that ω has the variable symbol ξ embedded somewhere within it as a term. If all variable symbols besides ξ in ω are quantified over in ω, then $(Q\xi)\omega$ is called a *closed wff* or *closed sentence*. In all of our applications, wffs will be closed.

Examples: $(\forall x)[P(x) \supset R(x)]$, $(\exists x)[P(x) \supset (\exists y)[R(x, y) \supset S(f(x))]]$

We would be able to show (after describing the semantics of quantifiers below) that $(\forall x)[(\forall y)\omega]$ is equivalent to $(\forall y)[(\forall x)\omega]$, so we could just as well group the universally quantified variables in a string in front of ω: $(\forall x, y)\omega$. In such a formula, ω is called the *matrix*. Similarly, for existential quantifiers. But mixtures of universal and existential quantifiers must retain their order! It is not the case that $(\forall x)[(\exists y)\omega]$ is equivalent to $(\exists y)[(\forall x)\omega]$.

2.13 Semantics of Quantifiers:

Universal Quantifiers:

$(\forall\xi)\omega(\xi)$ has the value *True* (under a given assignment of object constants, function constants, and relation constants to objects, functions, and relations) just in case $\omega(\xi)$ has the value *True* for *all* assignments of the variable symbol ξ to objects in the domain.

Existential Quantifiers:

$(\exists\xi)\omega(\xi)$ has the value *True* (under a given assignment to objects, functions, and relations) just in case $\omega(\xi)$ has the value *True* for *at least one* assignment of the variable symbol ξ to objects in the domain.

Rules of Inference:

The propositional-calculus rules of inference, suitably generalized, can also be used with the predicate calculus. These include *modus ponens*, $\wedge$ introduction and elimination, $\vee$ introduction, $\neg$ elimination, and resolution.

In addition to the propositional rules, twoimportant are

- *Universal instantiation (UI).* From $(\forall\xi)\omega(\xi)$, infer $\omega(\alpha)$, where $\omega(\xi)$ is any wff with variable ξ, α is any constant symbol, and $\omega(\alpha)$ is $\omega(\xi)$ with α substituted for ξ throughout ω.

  Example: From $(\forall x)P(x, f(x), B)$ infer $P(A, f(A), B)$.

- *Existential generalization (EG).* From $\omega(\alpha)$, infer $(\exists\xi)\omega(\xi)$, where $\omega(\alpha)$ is a wff containing a constant symbol α, and $\omega(\xi)$ is a form with ξ replacing every occurrence of α throughout ω.

  Example: From $(\forall x)Q(A, g(A), x)$ infer $(\exists y)(\forall x)Q(y, g(y), x)$.

It is easy to show that both UI and EG are sound rules of inference.

2.14 Resolution in the Predicate Calculus:

Unification:

abbreviate wffs of the form $(\forall\xi_1, \xi_2, \ldots, \xi_n)(\lambda_1 \vee \lambda_2 \vee \ldots \vee \lambda_k)$ by $\lambda_1 \vee \lambda_2 \vee \ldots \vee \lambda_k$,

where $\lambda_1, \lambda_2, \ldots, \lambda_k$ are literals that might contain occurrences of the variables $\xi_1, \xi_2, \ldots, \xi_n$. That is, I simply drop the universal quantifiers and assume universal quantification of any variables in the λ_i.

Wffs in this abbreviated form are called *clauses.*

write a clause using set notation $\{\lambda_1, \lambda_2, \ldots, \lambda_k\}$ and assume disjunction among the elements of the set.

If two clauses have matching but complementary literals, we can resolve them—just as in the propositional calculus. If we have a literal, $\lambda(\xi)$, in one clause (where ξ is a variable) and a complementary literal $\neg\lambda(\tau)$ in another clause, where τ is some term that does not contain ξ, we can substitute τ for ξ throughout the first clause and then do propositional resolution on the complementary literals to produce the *resolvent* of the two clauses.

The appropriate substitution is computed by a process called *unification.*

Unification is an extremely important process in AI.

UNIFY(Γ) (Γ is a set of list-structured expressions.)

1. $k \leftarrow 0, \Gamma_k \leftarrow \Gamma, \sigma_k \leftarrow \epsilon$ (Initialization step; ϵ is the empty substitution.)
2. If Γ_k is a singleton, exit with σ_k, the mgu of Γ. Otherwise, continue.
3. $D_k \leftarrow$ the disagreement set of Γ_k.
4. If there exists elements v_k and t_k in D_k such that v_k is a variable that does not occur in t_k, continue. Otherwise, exit with failure; Γ is not unifiable.
5. $\sigma_{k+1} \leftarrow \sigma_k\{t_k/v_k\}, \Gamma_{k+1} \leftarrow \Gamma_k\{t_k/v_k\}$ (Note that $\Gamma_{k+1} = \Gamma_k\sigma_{k+1}$.)
6. $k \leftarrow k + 1$
7. Go to step 2.

2.15 Predicate Calculus Resolution:

Suppose that γ_1 and γ_2 are two clauses (represented as sets of literals). If there is an atom ϕ in γ_1 and a literal $\neg\psi$ in γ_2 such that ϕ and ψ have a most general unifier, μ, then these two clauses have a resolvent, ρ. The resolvent is obtained by applying the substitution μ to the union of γ_1 and γ_2, leaving out the complementary literals. That is,

$$\rho = [(\gamma_1 - \{\phi\}) \cup (\gamma_2 - \{\neg\psi\})]\mu$$

Before two clauses are resolved, in order to avoid confusion about variables, we rename the variables in each clause so that none of the variables in one clause occurs in the other. For example, suppose we are resolving $P(x) \vee Q(f(x))$ against $R(g(x)) \vee \neg Q(f(A))$. We first rewrite the second clause, say, as $R(g(y)) \vee \neg Q(f(A))$ and then perform the resolution to obtain $P(A) \vee R(g(y))$. Renaming the variables is called *standardizing the variables apart.*

Completeness and Soundness

Predicate-calculus resolution is sound. That is, if ρ is the resolvent of two clauses ϕ and ψ, then $\{\phi, \psi\} \models \rho$. The proof of this fact is not much more difficult than the proof of soundness of propositional resolution. Just as in the propositional calculus, completeness of resolution is a more complicated matter. We cannot infer by resolution alone all the formulas that are logically entailed by a given set. For example, we cannot infer $P(A) \vee P(B)$ from $P(A)$. In propositional resolution, we surmounted this difficulty by using resolution refutation, and we can do the same in the predicate calculus. To guarantee refutation completeness, however, we must use the stronger rule of resolution just given.

Converting Arbitrary wffs to Clause Form

Just as in the propositional calculus, any wff can be converted to clause form. The steps are as follows:

1. Eliminate implication signs (same as in propositional calculus).
2. Reduce scopes of negation signs (same as in propositional calculus).
3. Standardize variables. Since variables within the scopes of quantifiers are like "dummy variables," they can be renamed so that each quantifier has its own variable symbol.

   Example: $(\forall x)[\neg P(x) \vee (\exists x)Q(x)]$ would be rewritten as $(\forall x)[\neg P(x) \vee (\exists y)Q(y)]$.
4. Eliminate existential quantifiers.

Example: In $(\forall x)[(\exists y)\mathtt{Height}(x, y)]$, the existential quantifier is within the scope of a universal quantifier, and thus the y that "exists" might depend on the value of x. For example, if the intended meaning of $(\forall x)[(\exists y)\mathtt{Height}(x, y)]$ is "every person, *x*, has a height, *y*," then obviously height depends on the person. Let this dependence be explicitly defined by some unknown function, say, $\mathtt{h}(x)$, which maps each value of x into the y that exists. Such a function is called a *Skolem function.*[1] If we use the Skolem function in place of the y that exists, we can eliminate the existential quantifier altogether and write $(\forall x)\mathtt{Height}[x, \mathtt{h}(x)]$.

The general rule for eliminating an existential quantifier from a wff is to replace each occurrence of its existentially quantified variable by a Skolem function whose arguments are those universally quantified variables that are bound by universal quantifiers whose scopes include the scope of the existential quantifier being eliminated. Function symbols used in Skolem functions must be "new" in the sense that they cannot be ones that already occur in any of the wffs to be used in a resolution refutation.

Thus, we can eliminate the $(\exists z)$ from

$[(\forall w)Q(w)] \supset (\forall x)\{(\forall y)\{(\exists z)[P(x, y, z) \supset (\forall u)R(x, y, u, z)]\}\}$ to yield

$[(\forall w)Q(w)] \supset (\forall x)\{(\forall y)[P(x, y, g(x, y)) \supset (\forall u)R(x, y, u, g(x, y))]$

and we can eliminate the $(\exists w)$ from

$(\forall x)\{\neg P(x) \vee \{(\forall y)[\neg P(y) \vee P(f(x, y))] \wedge (\exists w)[Q(x, w) \wedge \neg P(w)]\}\}$ to yield

$^*(\forall x)\{\neg P(x) \vee \{(\forall y)[\neg P(y) \vee P(f(x, y))] \wedge [Q(x, h(x)) \wedge \neg P(h(x))]\}\}$

where $g(x, y)$ and $h(x)$ are Skolem functions.

If the existential quantifier being eliminated is not within the scope of any universal quantifiers, we use a Skolem function of no arguments, which is just a constant. Thus, $(\exists x)P(x)$ becomes $P(Sk)$, where the constant symbol Sk is used to refer to the entity that we know exists. Again, it is necessary that Sk be a new constant symbol and not one used in other formulas to refer to known entities.

To eliminate all of the existentially quantified variables from a wff, we use the preceding procedure on each subformula in turn. Eliminating the existential quantifiers from a set of wffs produces what is called the *Skolem form* of the set of formulas.

It should be noted that the Skolem form of a wff is not *equivalent* to the original wff! The formula $(\exists x)P(x)$ is logically entailed by its Skolem form, $P(Sk)$, but not vice versa. As another example, note that $[P(A) \vee P(B)] \models (\exists x)P(x)$, but $[P(A) \vee P(B)] \not\models P(Sk)$. What is true is that a set of formulas, Δ, is satisfiable if and only if the Skolem form of Δ is. Or, more usefully for purposes of resolution refutations, Δ is unsatisfiable if and only if the Skolem form of Δ is unsatisfiable [Loveland 1978, pp. 41ff].

5. Convert to prenex form. At this stage, there are no remaining existential quantifiers, and each universal quantifier has its own variable symbol. We may now move all of the universal quantifiers to the front of the wff and let the scope of each quantifier include the entirety of the wff following it. The resulting wff is said to be in *prenex* form. A wff in prenex form consists of a string of quantifiers called a *prefix* followed by a quantifier-free formula called a *matrix*. The prenex form of the example wff marked with an * earlier is

$$(\forall x)(\forall y)\{\neg P(x) \vee \{[\neg P(y) \vee P(f(x, y))] \wedge [Q(x, h(x)) \wedge \neg P(h(x))]\}\}$$

6. Put the matrix in conjunctive normal form. Same as in propositional calculus. We may put any matrix into conjunctive normal form by repeatedly using one of the distributive rules, namely, by replacing expressions of the form $\omega_1 \vee (\omega_2 \wedge \omega_3)$ by $(\omega_1 \vee \omega_2) \wedge (\omega_1 \vee \omega_3)$.

   When the matrix of the preceding example wff is put in conjunctive normal form, it becomes

$$(\forall x)(\forall y)\{[\neg P(x) \vee \neg P(y) \vee P(f(x, y))] \wedge [\neg P(x) \vee Q(x, h(x))] \wedge [\neg P(x) \vee \neg P(h(x))]\}$$

7. Eliminate universal quantifiers. Since all of the variables in the wffs we use must be within the scope of a quantifier, we are assured that all the variables remaining at this step are universally quantified. Furthermore, the order of universal quantification is unimportant, so we may eliminate the explicit occurrence of universal quantifiers and assume, by convention, that all variables in the matrix are universally quantified. We are left now with just a matrix in conjunctive normal form.
8. Eliminate $\wedge$ symbols. We may now eliminate the explicit occurrence of $\wedge$ symbols by replacing expressions of the form $(\omega_1 \wedge \omega_2)$ with the set of wffs $\{\omega_1, \omega_2\}$. The result of repeated replacements is to obtain a finite set of wffs, each of which is a disjunction of literals. Any wff consisting solely of a disjunction of literals is called a clause. Our example wff is transformed into the following set of clauses:

$$\neg P(x) \vee \neg P(y) \vee P[f(x, y)]$$

$$\neg P(x) \vee Q[x, h(x)]$$

$$\neg P(x) \vee \neg P[h(x)]$$

9. Rename variables. Variable symbols may be renamed so that no variable symbol appears in more than one clause. We note that $(\forall x)[P(x) \wedge Q(x)]$ is equivalent to $[(\forall x)P(x) \wedge (\forall y)Q(y)]$. Our clauses are now

$\neg P(x1) \vee \neg P(y) \vee P[f(x1, y)]$

$\neg P(x2) \vee Q[x2, h(x2)]$

$\neg P(x3) \vee \neg P[h(x3)]$

The literals of a clause may contain variables, but these variables are always understood to be universally quantified.

2.16 Rule-Based Expert Systems:

One of the most successful applications of AI reasoning techniques using facts and rules has been in building *expert systems* that embody knowledge about a specialized field of human endeavor, such as medicine, engineering, or business.

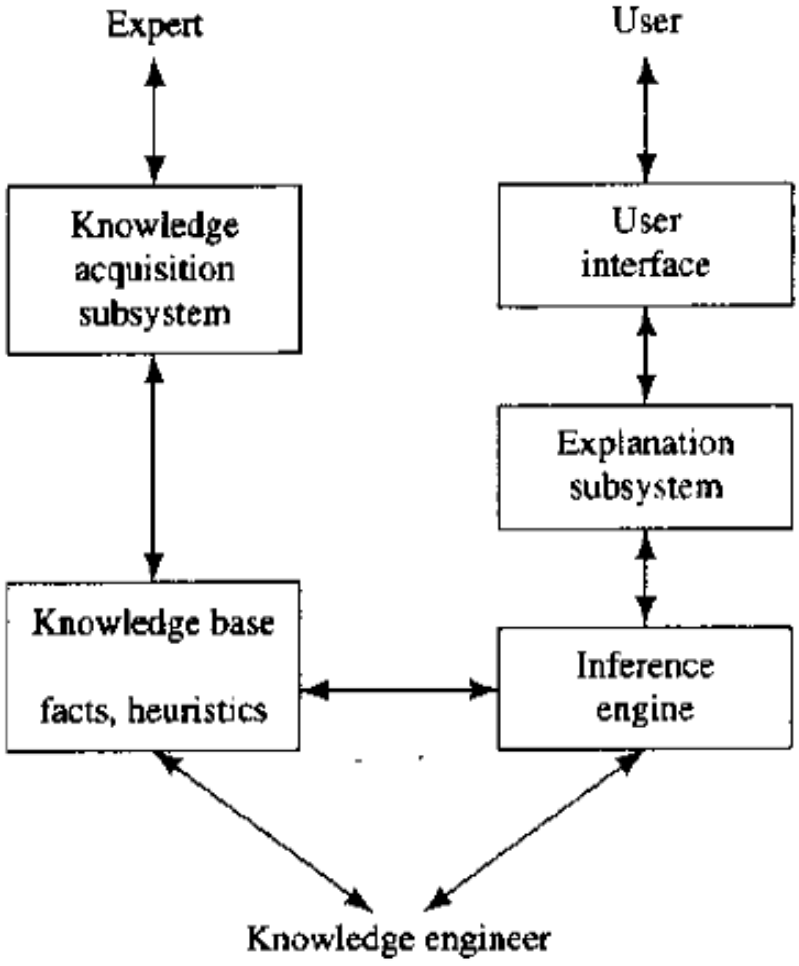


Figure 17.7

Basic Structure of an Expert System

AI programs that achieve expert-level competence in solving problems by bringing to bear a body of knowledge are called knowledge-based systems or expert systems. Often, the term expert systems is reserved for programs whose knowledge base contains the knowledge used by human experts, in contrast to knowledge gathered from textbooks or nonexperts. More often than not, the two terms—expert system and knowledge-based system—are used synonymously.

These four parts of the system are the parts that constitute the system as it is used in an application. In the construction of an expert system, a "knowledge engineer" (usually a computer scientist with AI training) works with an expert (or experts) in the field of application in order to represent the relevant knowledge of the expert in a form that can be entered into the knowledge base. This process is often aided by a knowledge acquisition subsystem that, among other things, checks the growing knowledge base for possible inconsistencies and incomplete information. These are then presented to the expert for resolution.

Rule-based expert systems are often based on reasoning with propositional logic Horn clauses (perhaps with some kind of additional mechanism for dealing with uncertainty). The knowledge base consists of rules gathered from experts. These systems have been applied in many settings. One could imagine a loan officer in a bank, for example, using such a system to help decide whether or not to grant a personal loan to an individual. A practical loan-approval system would take into account many factors, many more than we would want to consider in an illustrative example. But I can give an approximate idea of how such a system might work by describing a vastly oversimplified version. Suppose the following atoms are intended to denote the associated propositions:

2.17 Representing Common Sense Knowledge:

Much more extensive knowledge will be required for truly versatile, human-level AI systems. And these systems will need to know about many topics that have proved difficult to conceptualize formally. It is perhaps paradoxical that AI scientists find the very subjects that are easy for ten-year-old humans more refractory to AI methods than are subjects that experts must study for years. Physicists are able to describe detailed, exact physical phenomena by wave equations, relativity theory, and other mathematical constructs, but AI researchers still argue about the best way to represent the simple (but very useful) facts that a liquid fills the shape of a cup and will fall out if the cup is turned upside down. The highly theoretical characterizations invented by physicists and mathematicians seem easier to formalize than do ideas that, after all, enabled human beings to function pretty well even before Aristotle.

Consider just some of the things that a ten-year-old knows:

If you drop an object, it will fall. (Nowadays, a ten-year-old might also know that objects wouldn't "fall" if dropped in an orbiting satellite.)

People don't exist before they are born.

Fish live in water and will die if taken out.

People buy bread and milk in a grocery store.

People typically sleep at night.

Knowledge of this sort is usually called *commonsense knowledge*. Typically, knowledge about any subject is spread over a variety of levels—ranging from that possessed by the person-in-the-street to that of the specialist. But the most advanced scientific theories extant around 500 B.C., say, were little more than careful verbal formulations of people's everyday observations. To some the earth was flat, objects fell to the earth because that was "their natural place," and human diseases were caused by a variety of colorful "influences." When knowledge was scarce, little separated everyday, commonsense knowledge from advanced scientific knowledge. Commonsense knowledge was (and still is) adequate for many of the things that humans want to do. Scientific knowledge gradually separated itself from commonsense knowledge as people sought more precise descriptions of their world.

Difficulties in Representing Commonsense Knowledge

What are some of the reasons that it has proved difficult to formalize commonsense knowledge? Probably one reason is its sheer bulk. Much expert knowledge can be compartmentalized in such a way that a few hundred or a few thousand facts are sufficient to build useful expert systems. How many will be needed by a system capable of general human-level intelligence? No one knows for sure. Doug Lenat, who is bravely leading an effort to build a large knowledge base of such facts, called CYC, thinks that between one and ten million will be needed In 1990, the CYC authors said:

> Perhaps the hardest truth to face, one that AI has been trying to wriggle out of for 34 years, is that there is probably no elegant, effortless way to obtain this immense knowledge base. Rather, the bulk of the effort must (at least initially) be manual entry of assertion after assertion.

> Another difficulty is that commonsense knowledge seems not to have well-defined frontiers that enable us to get a grip on parts of it independently of the other parts. Conceptualizations of the commonsense world would probably involve many entities, functions, and relations that would crop up diffusely throughout the conceptualization. Thus, as we attempt to develop a conceptualization, we wouldn't know whether we had "gotten it right" until we had nearly finished the entire, exceedingly large job.