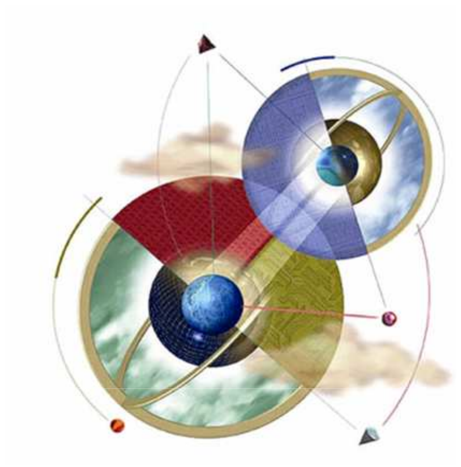


Chapter 3: Methods of Inference

Expert Systems: Principles and Programming, Fourth Edition

Objectives

- Learn the definitions of trees, lattices, and graphs
- Learn about state and problem spaces
- Learn about AND-OR trees and goals
- Explore different methods and rules of inference
- Learn the characteristics of first-order predicate logic and logic systems

Objectives

- Discuss the resolution rule of inference, resolution systems, and deduction
- Compare shallow and causal reasoning
- How to apply resolution to first-order predicate logic
- Learn the meaning of forward and backward chaining

Objectives

- Explore additional methods of inference
- Learn the meaning of Metaknowledge
- Explore the Markov decision process

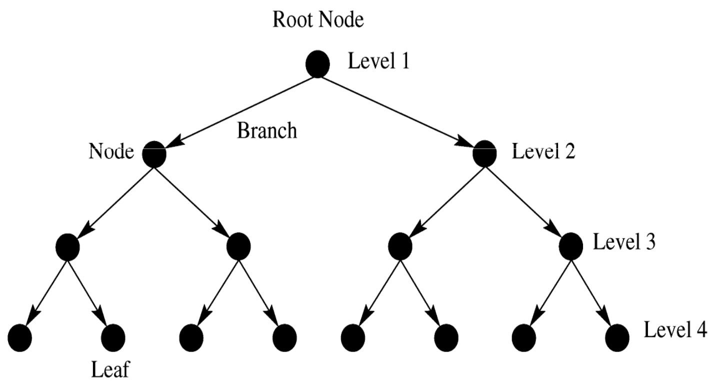
Trees

- A tree is a hierarchical data structure consisting of:
  - Nodes – store information or knowledge
  - Branches – connect the nodes
- The top node is the root, occupying the highest hierarchy.
- The leaves are at the bottom, occupying the lowest hierarchy.

Trees

- Every node, except the root, has exactly one parent. (Otherwise it becomes a graph.)
- Every node may give rise to zero or more child nodes.
- A binary tree restricts the number of children per node to a maximum of two.
- A degenerate tree is a tree where each parent has only one child (behaving like a linked list having only a single pathway from root to its one leaf).

Figure 3.1 Binary Tree



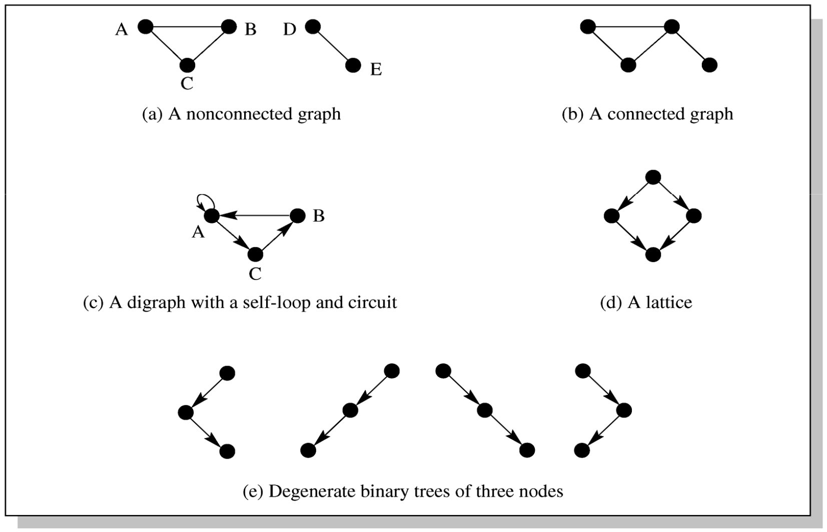
Graphs

- Trees are a special case of a graph.
- Graphs are sometimes called a network or net.
- A graph can have zero or more links between any pair of nodes – there is no distinction between parent and child.
- A map is an example of a graph, where the cities are nodes and the roads are links.
- Sometimes links have weights – **weighted graph**; or, arrows – **directed graph**.
  - E.g. weight: distance & arrow : direction

Graphs

- **Simple graphs** are unweighted, undirected graphs containing no **loops**, or links that come back onto the node itself.
- **A circuit (cycle)** is a path through the graph beginning and ending with the same node.
- **Acyclic graphs** have no cycles.
- **Connected graphs** have a path from any node to any other node in the graph.
- Digraph is a short for directed graph.
- **Lattice** is a directed acyclic graph.

Figure 3.2 Typical Graphs


A
B
C
D
E
(a) A nonconnected graph
(b) A connected graph
A
B
C
(c) A digraph with a self-loop and circuit
(d) A lattice
(e) Degenerate binary trees of three nodes

Making Decisions

- Trees / lattices are useful for classifying objects in a hierarchical nature.
  - E.g. family trees show the relationships.
- Trees / lattices are useful for making decisions and simple reasoning.
- We refer to trees / lattices as structures.
- Decision trees are structures useful for representing and reasoning about knowledge.

Binary Decision Trees

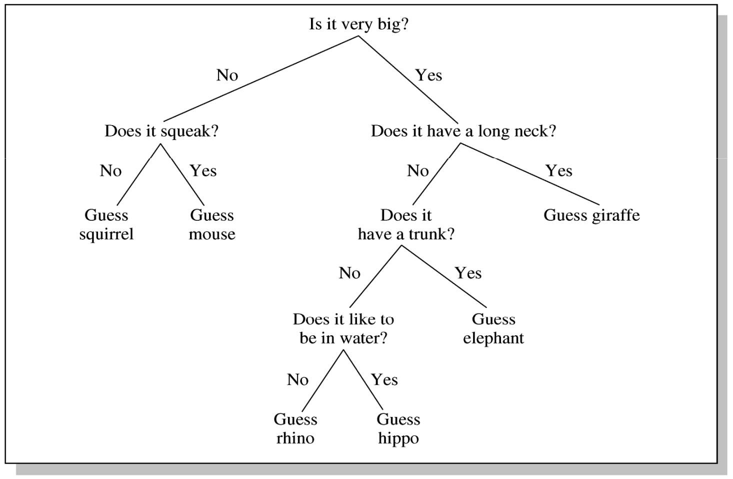
- It is easy to construct and is very efficient.
- Allows binary decisions (two responses).
- A binary decision tree having *N* nodes:
  - All internal nodes are questions.
  - The two branches are “yes” or “no” responses.
  - All leaves will be final answers.
  - *N* questions can lead to an answer out of maximally 2^N possible answers.
  - E.g. ten questions can classify one of 1024 animals (refer to Fig. 3.3).

Binary Decision Trees

- Decision trees can be self learning.
  - If a guess is wrong, the user can be queried to add new and correct question and answers.
  - New node, branches and leaves can be created and added to the tree dynamically.
  - E.g. Add hamster to Fig. 3.3.
- Decision trees can be translated into production rules by breadth-first search of the tree and generating IF-THEN rule at each node.
  - If question="Is it very big?" and answer="No" then question:="Does it squeak?"

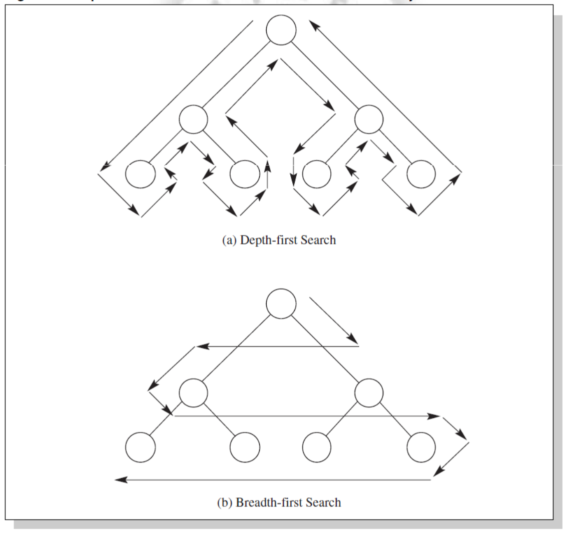
Decision Tree Example

Figure 3.3 Decision Tree Showing Knowledge About Animals



Searching the Decision Tree

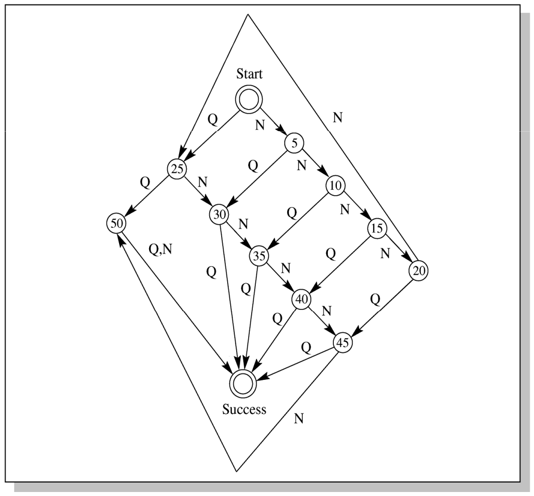
Figure 2.7 Depth-First and Breadth-First Searches for an Arbitrary Tree



State and Problem Spaces

- An application of the graph technique.
- A state space can be used to define the behavior of an object (system).
- Each node corresponds to a state – a collection of characteristics that define the status of the object.
- The links show the transitions the object can make from one state to another.

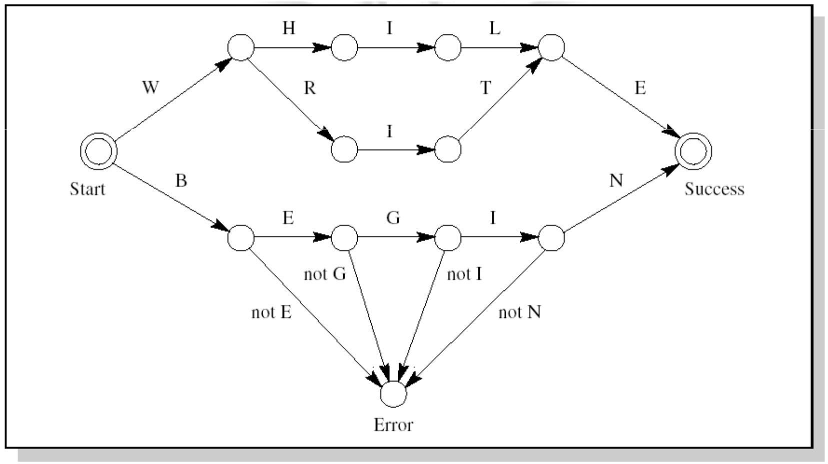
Figure 3.5 State Diagram for a Soft Drink Vending Machine Accepting Quarters (Q) and Nickels (N) Note: The price for the drink is 55 cents. For simplicity other coins are not included.



Finite State Machine

- A FSM is a kind of state space describing the finite number of states of a machine (system).
- At any one time, the machine is in one particular state.
- The machine accepts input and progresses to the next state. A good design should consider invalid inputs and provide for transition to error states.
- FSMs are often used in compilers and validity checking programs.

Fig. 3.6 Part of a Finite State Machine for Determining Valid Strings WHILE, WRITE and BEIGIN Note: Only some of the error links are shown



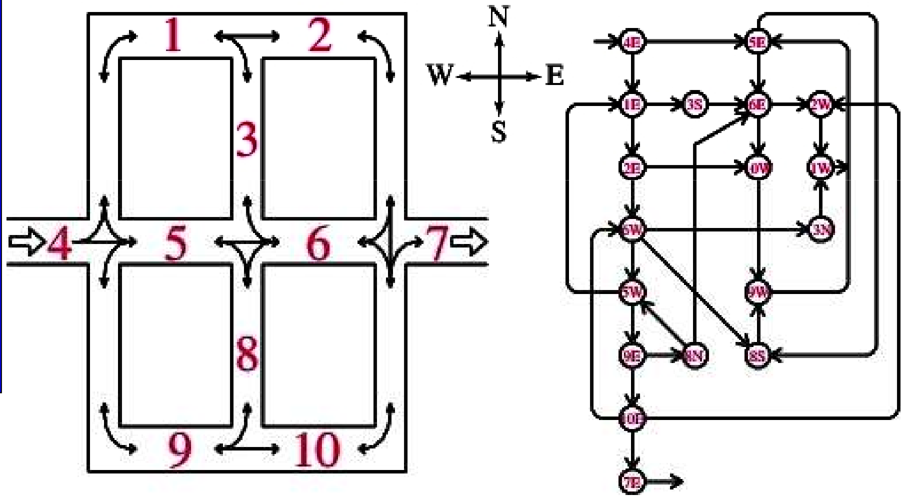
Using FSM to Solve Problems

- A state diagram (FSM) showing how to solve a problem is also called a problem space.
  - Nodes: intermediate stages in problem solving
  - Links: intermediate steps in the solution
  - Valid path(s) from the start state to the success state(s) is (are) the solution(s).

Problem Space Example

- A car must traverse a town while obeying all traffic laws, and making no U-turns. Initially, the car is at position 4, travelling east, and has a choice of moving to position 1, travelling east, or position 5, travelling east. The state diagram corresponding to the maze is illustrated in the right figure (See next page).

Problem Space Example



AND-OR Trees and Goals

- Many expert systems use backward chaining to find solutions to problems, e.g. PROLOG.
- PROLOG uses backward chaining to divide a problem into smaller problems and then solves them.
- AND-OR trees / lattices are useful for solving backward chaining problems (Fig. 3.9).
- AND-OR trees / lattices can use logic gates to describe problems making hardware implementation for fast processing speed possible (Figs. 12 & 13)

Figure 3.9 AND-OR Lattice Showing How to Obtain a College Degree

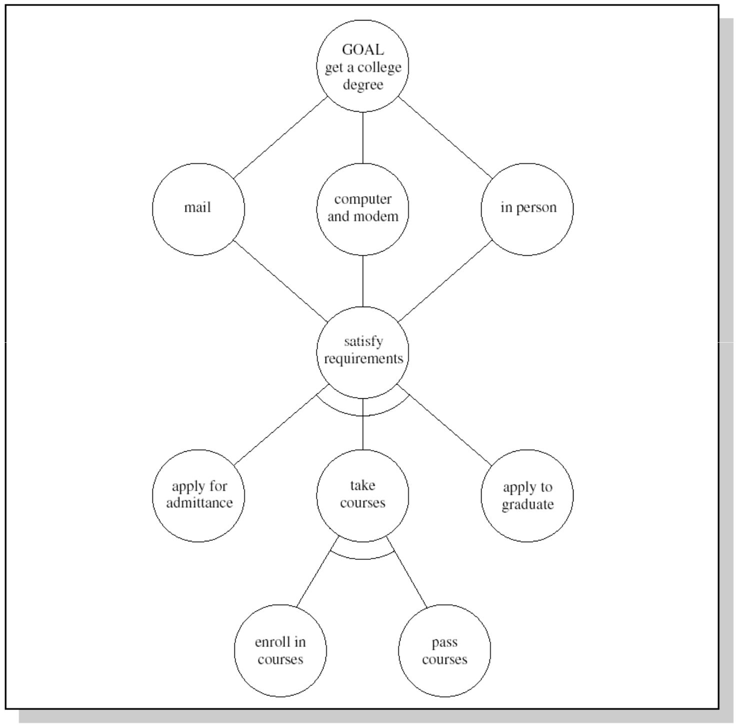


Figure 3.12 AND, OR, and NOT Logic Gate Symbols

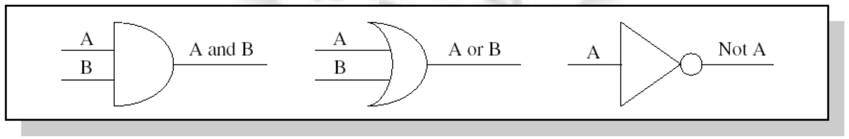
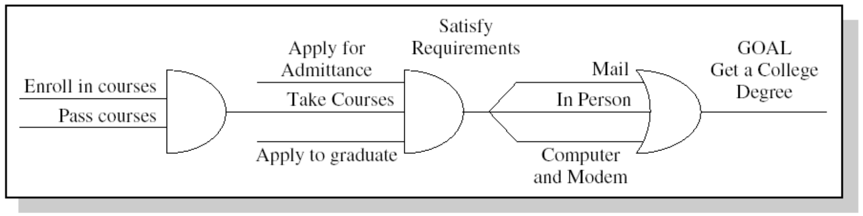


Figure 3.13 AND-OR Logic Gate Representation for Figure 3.9



Types of Inference

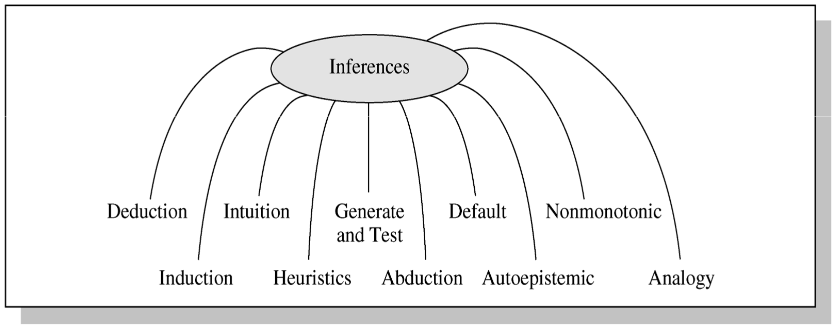
Logic can be used to represent knowledge as well as making references about the it.

- Deduction – reasoning where conclusions must follow from premises
- Induction – inference is from the specific case to the general
- Intuition – no proven theory and unconscious (expert system does not implement this)
- Heuristics – rules of thumb based on experience
- Generate and test – trial and error
- Default – assume common knowledge in absence of specific one

Types of Inference

- Abduction – reasoning back from a true condition to the premises that may have caused the condition (making a best guess)
- Autoepistemic – rely on self-knowledge and any unknown fact is assumed to be false
- Nonmonotonic – allowing new knowledge that contradicts the previous ones to be added
- Analogy – inferring conclusions based on similarities with other situations (heuristic)
- Commonsense –reasoning that people use in everyday life; a combination of all above (very difficult for computers)

Figure 3.14 Types of Inference



Deductive Logic

- Most often used method of reasoning
- Based on a chain of reasoning in some form
  - Conclusions reached by following true premises and the forms must themselves be true
- Argument – logical reasoning with group of statements where the last is justified on the basis of the previous ones
- Syllogism – one type of deductive argument which has two premises and one conclusion

Syllogisms vs. Rules

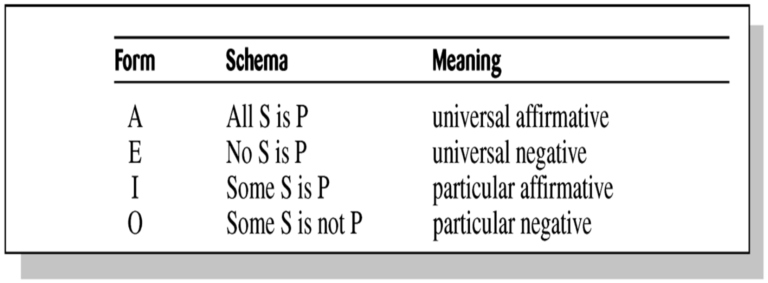
- Syllogism:
  - All basketball players are tall.
  - Jason is a basketball player.
  - ∴ Jason is tall.
- Syllogism can be expressed as IF-THEN rule:

IF All basketball players are tall and
Jason is a basketball player
THEN Jason is tall.

Categorical Syllogism

In classic syllogism, premises and conclusions are defined using categorical statements of the following four forms:

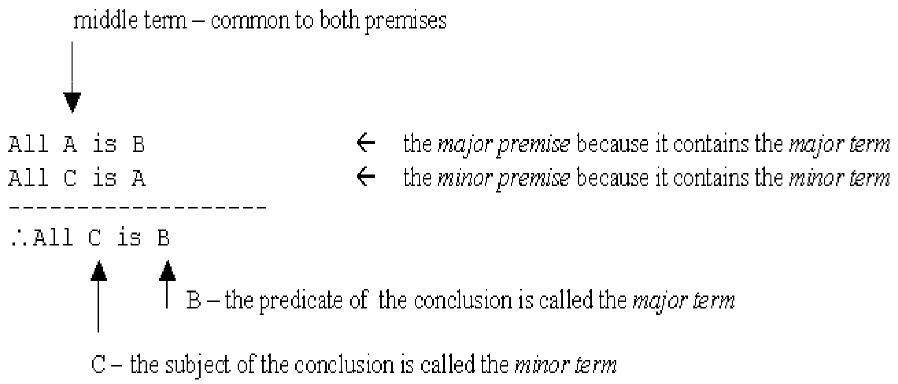
Table 3.2 Categorical Statements



Form	Schema	Meaning
A	All S is P	universal affirmative
E	No S is P	universal negative
I	Some S is P	particular affirmative
O	Some S is not P	particular negative

Categorical Syllogisms

Some terms in the syllogism:



middle term – common to both premises

```
All A is B           ← the major premise because it contains the major term
All C is A           ← the minor premise because it contains the minor term
-------------------
∴All C is B
```

B – the predicate of the conclusion is called the *major term*

C – the subject of the conclusion is called the *minor term*

Categorical Syllogisms

To write the syllogism in standard form, we would write:

```
Major Premise:    All A is B
Minor Premise:    All C is A
---------------------------------
Conclusion        ∴All C is B
```

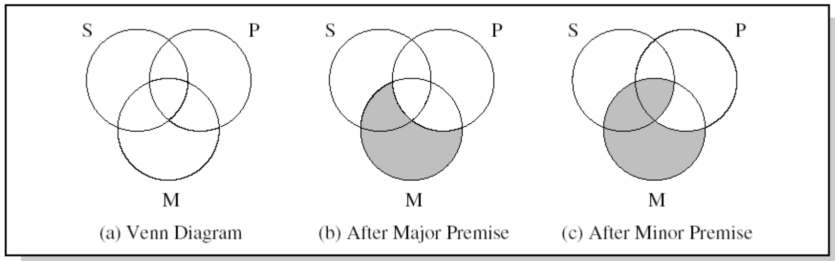
The validity of syllogistic arguments can be proved by Venn diagram.

Proving the Validity of Syllogistic Arguments Using Venn Diagrams

All M is P

<u>No S is M</u>

∴ No S is P



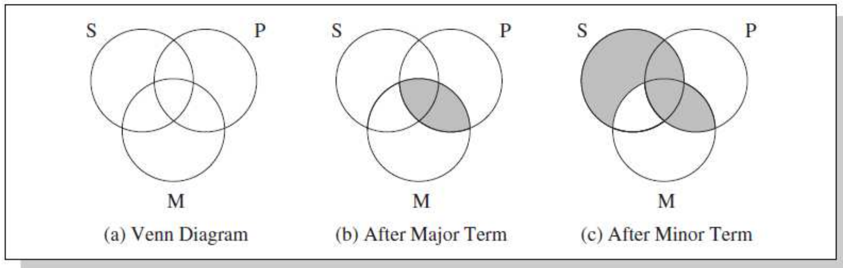
(False)

Proving the Validity of Syllogistic Arguments Using Venn Diagrams

No M is P

<u>All S is M</u>

∴ No S is P



(True)

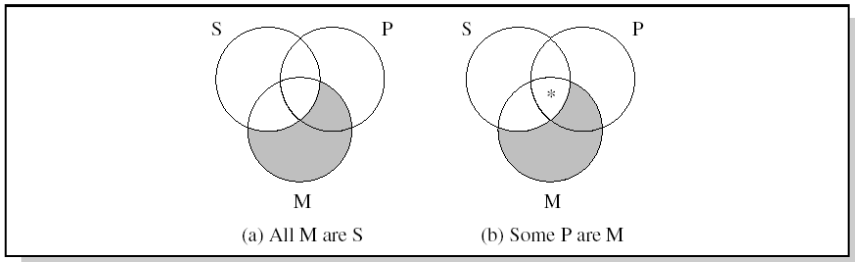
General Rules for Drawing Venn Diagrams

Rules to follow when “some” quantifiers are used:

1. If a class is empty, it is shaded.
2. Universal statements, A and E are always drawn before particular ones.
3. If a class has at least one member, mark it with an *.
4. If a statement does not specify in which of two adjacent classes an object exists, place an * on the line between the classes.
5. If an area has been shaded, no * can be put in it.

Proving the Validity of Syllogistic Arguments Using Venn Diagrams

Some P are M
<u>All M are S</u>
∴ Some S are P



(a) All M are S (b) Some P are M

(True)

Rules of Inference – Propositional Logic

- Venn diagrams are inconvenient for validity check for complex arguments.
- Syllogisms address only a small portion of the possible logical statements – only four possible forms (A, E, I , O).
- Propositional logic offers a more powerful way of describing arguments by allowing more complex types of statements.

Direct Reasoning Modus Ponens

```
If I am hungry, I will eat
I am hungry
----------------------------
∴ I will eat

A = I am hungry

B = I will eat

A → B
A
-------
∴ B
```

Truth Table Modus Ponens

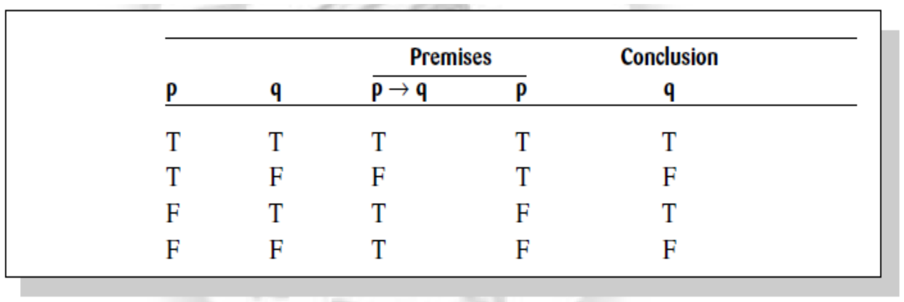
The validity of propositional logic can be proved by constructing a truth table and examine it for the tautology.

$p \to q$
<u>p .</u>
q

p	q	$p \to q$	$(p \to q) \wedge p$	$(p \to q) \wedge p \to q$
T	T	T	T	T
T	F	F	F	T
F	T	T	F	T
F	F	T	F	T

Truth Table Modus Ponens

A shorter method for determining the validity of the argument is to consider only rows in which the premises are all true. If the conclusions are also true, the argument is valid.



		Premises		Conclusion
p	q	$p \to q$	p	q
T	T	T	T	T
T	F	F	T	F
F	T	T	F	T
F	F	T	F	F

Truth Table Fallacy of Converse

If there are no bugs, then the program compiles
The program compiles
∴ There are no bugs

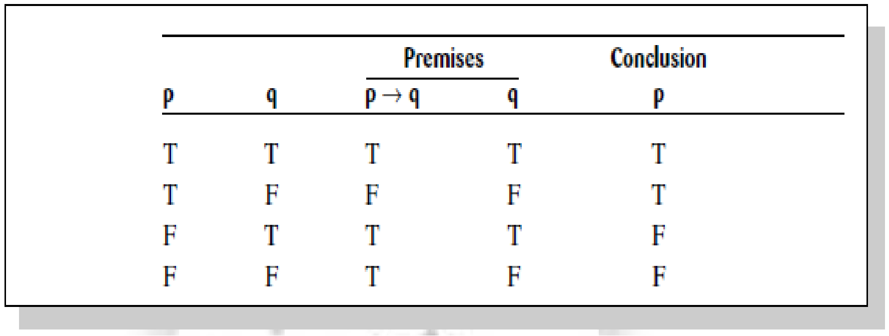
$p \to q$ p: there are no bugs
$\underline{q \quad .}$ q: the program compiles
$\therefore p$

p	q	$p \to q$	$(p \to q) \wedge q$	$(p \to q) \wedge q \to p$
T	T	T	T	T
T	F	F	F	T
F	T	T	T	F
F	F	T	F	T

Truth Table Fallacy of Converse

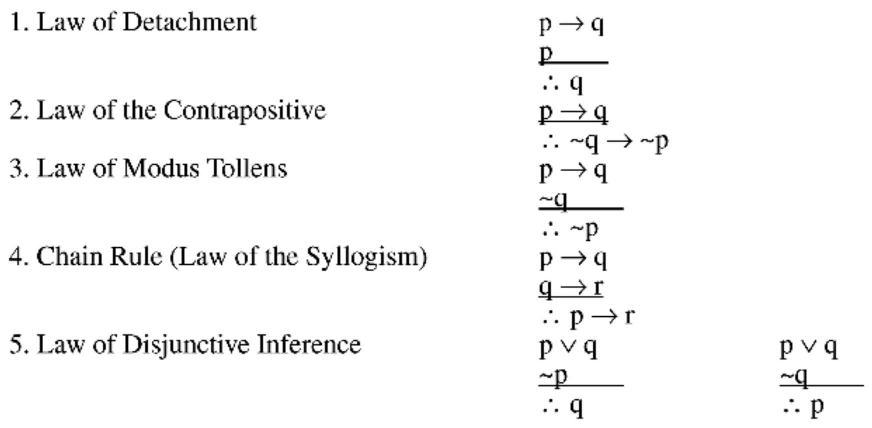
Short-form Truth Table for:

$p \to q$ p: there are no bugs
$\underline{q \quad .}$ q: the program compiles
$\therefore p$



		Premises		Conclusion
p	q	$p \to q$	q	p
T	T	T	T	T
T	F	F	F	T
F	T	T	T	F
F	F	T	F	F

Some Rules of Inference



1. Law of Detachment
$p \to q$
$\underline{p \qquad}$
$\therefore q$

2. Law of the Contrapositive
$\underline{p \to q}$
$\therefore \sim q \to \sim p$

3. Law of Modus Tollens
$p \to q$
$\underline{\sim q \qquad}$
$\therefore \sim p$

4. Chain Rule (Law of the Syllogism)
$p \to q$
$\underline{q \to r}$
$\therefore p \to r$

5. Law of Disjunctive Inference
$p \vee q$ / $\underline{\sim p \qquad}$ / $\therefore q$
$p \vee q$ / $\underline{\sim q \qquad}$ / $\therefore p$

Some Rules of Inference

6. Law of the Double Negation	$\begin{array}{l} \sim(\sim p) \\ \hline \therefore p \end{array}$	
7. De Morgan's Law	$\begin{array}{l} \sim(p \wedge q) \\ \hline \therefore \sim p \vee \sim q \end{array}$	$\begin{array}{l} \sim(p \vee q) \\ \hline \therefore \sim p \wedge \sim q \end{array}$
8. Law of Simplification	$\begin{array}{l} p \wedge q \\ \hline \therefore p \end{array}$	$\begin{array}{l} \sim(p \vee q) \\ \hline \therefore \sim q \end{array}$
9. Law of Conjunction	$\begin{array}{l} p \\ q \\ \hline \therefore p \wedge q \end{array}$	
10. Law of Disjunctive Addition	$\begin{array}{l} p \\ \hline \therefore p \vee q \end{array}$	
11. Law of Conjunctive Argument	$\begin{array}{l} \sim(p \wedge q) \\ p \\ \hline \therefore \sim q \end{array}$	$\begin{array}{l} \sim(p \wedge q) \\ q \\ \hline \therefore \sim p \end{array}$

Using Rules of Inference

Chip prices rise only if the yen rises.
The yen rises only if the dollars falls and
if the dollar falls then the yen rises.
Since chip prices have risen, then dollar must
have fallen.

(Note: X only if Y ≡ if X then Y)

Let:

C = chip prices rise
Y = yen rises
D = dollar falls

Using Rules of Inference

Then:

1) $C \rightarrow Y$
2) $(Y \rightarrow D) \wedge (D \rightarrow Y)$
3) $\underline{C \qquad\qquad\qquad .}$
$\therefore D$

Proof:

4)	$Y \equiv D$	(2: equivalence)
5)	$C \rightarrow D$	(1: substitution)
6)	D	(5, 3: Modus Ponens)

Note: $(p \rightarrow q) \wedge (q \rightarrow p)$ means $p \leftrightarrow q$. If $p \leftrightarrow q$, then $p \equiv q$, which means p and q take the same truth value in all cases.

Using Rules of Inference

1) $A \vee B$
2) $C \wedge D$
3) $A \rightarrow \sim D$
4) $\underline{B \rightarrow X \qquad\qquad\qquad .}$
$\therefore X$

Proof:

5)	D	(2: simplification)
6)	$\sim A$	(3, 5: modus tollens)
7)	B	(1, 6: law of disjunctive inf.)
8)	X	(4, 7: modus ponens)

Limitations of Propositional Logic

- Propositional logic can not prove the validity of syllogistic arguments containing quantifiers (all, none, some, etc.), because it does not examine the internal structure of propositions.
- For example:

All men are mortal	p
Socrates is a man	q .
Therefore, Socrates is mortal	∴ r

Limitations of Propositional Logic

- A simple way to make propositional logic work is to remove all quantifiers:
- For example:

  All men are mortal
  Socrates is a man
  Therefore, Socrates is mortal

- ⇓

  If Socrates is a man, then Socrates is mortal
  Socrates is a man
  Therefore, Socrates is mortal

Limitations of Propositional Logic

- In some cases, we may be lucky to prove the validity of the argument.

If **a**ll the committee members vote for the bill, then the bill will **p**ass.
It is not the case that some members will not vote for the bill.
So, the bill will pass.

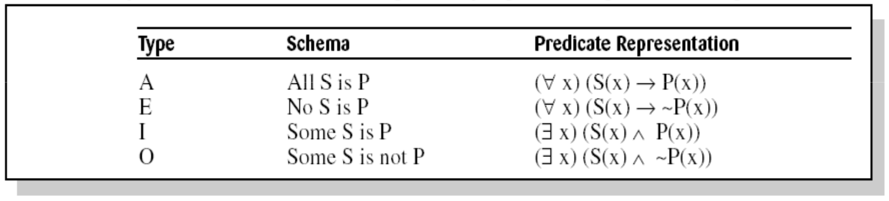
$A \rightarrow P$
$\underline{\sim(\sim A)}$
P

First-Order Predicate Logic

- The limitation of propositional logic can be overcome by using predicate logic, which is a superset of propositional logic.
- Syllogistic logic can be completely described by predicate logic.
- The Rule of Universal Instantiation states that an individual may be substituted for a universe.

First-Order Predicate Logic

Table 3.12 Representation of the Four Categorical Syllogisms Using Predicate Logic



Type	Schema	Predicate Representation
A	All S is P	$(\forall x)(S(x) \rightarrow P(x))$
E	No S is P	$(\forall x)(S(x) \rightarrow \sim P(x))$
I	Some S is P	$(\exists x)(S(x) \land P(x))$
O	Some S is not P	$(\exists x)(S(x) \land \sim P(x))$

First-Order Predicate Logic

All men are mortal

Socrates is a man

Therefore, Socrates is mortal

Let $H(x) \equiv$ x is a man, $M(x) \equiv$ x is mortal, and $s \equiv$ Socrates

1. $(\forall x)(H(x) \rightarrow M(x))$
2. $H(s)$
3. $H(s) \rightarrow M(s)$ (1: Universal Instantiation)
4. $M(s)$ (2, 3: Modus, Ponens)

Logic Systems

- A logic system (e.g. expert system) is a collection of objects such as rules, axioms, statements, and so forth in a consistent manner.
- Each logic system relies on formal definitions of its axioms (postulates) which are fundamental definitions of the system.
- An axiom is simply a fact or assertion that cannot be proven from within the system.
- From axioms, it can be determined what can be proven.

Goals of a Logic System

- Be able to specify the forms of statements – well formulated formulas (wffs).
  - E.g. All S is P $p \rightarrow q$
- Indicate the rules of inference that are valid.
  - E.g. $p \rightarrow q$

    $\underline{p \qquad .}$

    q
- Extend itself by discovering new wffs (theorems) and new rules of inference that are valid and thereby extend the range of arguments that can be proven.

Requirements of a Formal System

1. An alphabet of symbols (e.g. propositions).
2. A set of finite strings of these symbols, the wffs (e.g. a statement: if P_1 then P_2).
3. Axioms, the definitions of the system (e.g. propositions that are known facts).
4. Rules of inference, which enable a wff to be deduced as the conclusion of a finite set of other wffs – axioms or other theorems of the logic system.

Requirements of a Formal System

5. Completeness – every wff can either be proved or refuted.
6. The system must be sound – every theorem is a logically valid wff.

Simple example (a unary system of odd numbers):

Alphabet: “1” (only one symbol)

Axiom: “1” (happen to be the same as the symbol 1)

Rule of Inference: If string \$ is a theorem, then so is the string \$11, or \$ $\rightarrow$ \$11

wffs = {1, 111, 11111, 1111111, …}

Resolution

- Commonly used in theorem-proving (not deriving a new theorem)
  - Production rules and resolution systems are two popular paradigms for proving theorems
  - Using rules of inference (modus ponens, modus tollens, chaining, etc.) may take a lot of trial-and-errors
- A simple, practical, yet powerful technique
  - Making it easier to build a mechanical system such as PROLOG which uses one general-purpose inference rule of resolution rather than systems that attempt to implement many different rules of inference

Resolution

- Resolution is based on representing rules in normal forms (consisting of $\vee$, $\wedge$ and ~ only).
- Full clausal form $\rightarrow$ normal form:
  - Because : $p \rightarrow q \equiv \sim p \vee q$ $\quad\sim(p \wedge q) \equiv \sim p \vee \sim q$
  - $\therefore A_1 \wedge A_2 \wedge \ldots A_n \rightarrow B_1 \vee B_2 \vee \ldots B_m$

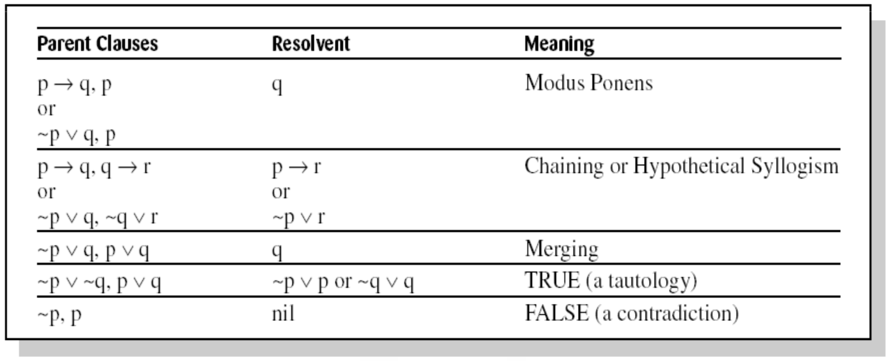
    $\equiv \sim(A_1 \wedge A_2 \wedge \ldots A_n) \vee (B_1 \vee B_2 \vee \ldots B_m)$

    $\equiv \sim A_1 \vee \sim A_2 \vee \ldots \sim A_n \vee B_1 \vee B_2 \vee \ldots B_m$
- PROLOG uses Horn clause, a restricted type of clausal form:
  - $A_1 \wedge A_2 \wedge \ldots A_n \rightarrow B$
  - B :- A_1, A_2, … A_n

Resolution

- The resolution method is applied to the normal forms, which operates on pairs of disjuncts to produce a new disjuncts and simplifies the normal forms.

Table 3.13 Clauses and Resolvents



Parent Clauses	Resolvent	Meaning
$p \rightarrow q, p$ or $\sim p \vee q, p$	q	Modus Ponens
$p \rightarrow q, q \rightarrow r$ or $\sim p \vee q, \sim q \vee r$	$p \rightarrow r$ or $\sim p \vee r$	Chaining or Hypothetical Syllogism
$\sim p \vee q, p \vee q$	q	Merging
$\sim p \vee \sim q, p \vee q$	$\sim p \vee p$ or $\sim q \vee q$	TRUE (a tautology)
$\sim p, p$	nil	FALSE (a contradiction)

Resolution

- Proof by resolution:

$$\begin{array}{lcl} A \rightarrow B & & \sim A \vee B \\ B \rightarrow C & \equiv & \sim B \vee C \\ \underline{A \quad\quad .} & & \underline{A \quad\quad .} \\ C & & C \end{array}$$

Finding the resolvent:

$(\sim A \vee B) \wedge (\sim B \vee C) \wedge A \equiv (\sim A \vee C) \wedge A \equiv C$

- In practice – proof by resolution contradiction:

If $(\sim A \vee B) \wedge (\sim B \vee C) \wedge A \equiv C$

then $(\sim A \vee B) \wedge (\sim B \vee C) \wedge A \wedge \sim C \equiv C \wedge \sim C \equiv \text{nil}$

Resolution

- Proof by resolution:

$$\begin{array}{lcl} A \rightarrow B & & \sim A \vee B \\ B \rightarrow C & \equiv & \sim B \vee C \\ \underline{C \rightarrow D} & & \underline{\sim C \vee D} \\ A \rightarrow D & & \sim A \vee D \end{array}$$

Finding the resolvent:

$(\sim A \vee B) \wedge (\sim B \vee C) \wedge (\sim C \vee D) \wedge \sim (\sim A \vee D) \equiv$

$(\sim A \vee B) \wedge (\sim B \vee C) \wedge (\sim C \vee D) \wedge A \wedge \sim D \equiv$

$(\sim A \vee C) \wedge (\sim C \vee D) \wedge A \wedge \sim D \equiv$

$(\sim A \vee D) \wedge A \wedge \sim D \equiv D \wedge \sim D \equiv \text{nil}$

$\therefore A \rightarrow D$ is valid

Shallow and Causal Reasoning

- In shallow reasoning, there is little/no causal chain of cause and effect from one rule to another
  - longer chain represents more causal or deep knowledge.
  - E.g. $A \rightarrow B, B \rightarrow C, C \rightarrow D \quad \therefore A \rightarrow D$
- Another major factor for shallow reasoning is due to the quality of knowledge
- Experiential knowledge is another term for shallow knowledge which is based on experience
- Advantage of shallow reasoning is ease of programming (smaller, faster, cheaper programs)

Shallow and Causal Reasoning

- Causal reasoning can be used to construct a model that behaves like the real system – can be used to answer “what if questions”
  - E.g. an expert system in medicine
- Causal models are not always necessary or desirable due to practical situation
  - E.g. MUD expert system serving as a consultant to drilling fluid or mud engineers
- Frames are used for causal / deep reasoning.

Shallow and Causal Reasoning

- Shallow reasoning example:

IF	a car has
	a good battery,
	good sparkplugs,
	gas,
	good tires
THEN	the car can move

Shallow and Causal Reasoning

- Deep reasoning example:

IF	the battery is good
THEN	there is electricity
IF	there is electricity and sparkplugs are good
THEN	the sparkplugs will fire
IF	the sparkplugs fire and there is gas
THEN	the engine will run
IF	the engine runs and there are good tires
THEN	the car will move

- Causal reasoning implies deep understanding of the subject – easier to determine what effect a bad component will have.

Resolution and First-Order Predicate Logic

Some programmers hate failures
<u>No programmer hates any success</u>
∴ No failure is a success

Let:

P(x) = x is a programmer	S(x) = x is a success
F(x) = x is a failure	H(x, y) = x hates y

Then premises and negated conclusion are:

1) $(\exists x)\ [P(x) \wedge (\forall y)\ (F(y) \rightarrow H(x, y))]$

2) $(\forall x)\ [P(x) \rightarrow (\forall y)\ (S(y) \rightarrow \sim H(x, y))]$

3) $\sim (\forall y)\ (F(y) \rightarrow \sim S(y))$

Conversion to the Normal Form

Nine steps for the conversion:

1. Eliminate conditionals.
   - 1) $\Rightarrow$ $(\exists x)\,[P(x) \wedge (\forall y)\,(\sim F(y) \vee H(x, y))]$
   - 2) $\Rightarrow$ $(\forall x)\,[\sim P(x) \vee (\forall y)\,(\sim S(y) \vee \sim H(x, y))]$
   - 3) $\Rightarrow$ $\sim (\forall y)\,(\sim F(y) \vee \sim S(y))$
2. When possible, eliminate negations or reduce their scope.
   - 3) $\Rightarrow$ $(\exists y)\,(F(y) \wedge S(y))$
3. Standardize variables (each quantifier has unique variable name).
   e.g. $(\exists x) \sim P(x) \vee (\forall x)\, P(x) \Rightarrow (\exists x) \sim P(x) \vee (\forall y)\, P(y)$

Conversion to the Normal Form

4. Eliminate existential quantifiers using Skolem functions.
   - 1) $\Rightarrow$ $P(a) \wedge (\forall y)\,(\sim F(y) \vee H(a, y))$
   - 2) $(\forall x)\,[\sim P(x) \vee (\forall y)\,(\sim S(y) \vee \sim H(x, y))]$
   - 3) $\Rightarrow$ $F(b) \wedge S(b)$
5. Convert wff to prenex form (move $\forall$ to front).
   - 1) $\Rightarrow$ $(\forall y)\,[P(a) \wedge (\sim F(y) \vee H(a, y))]$
   - 2) $\Rightarrow$ $(\forall x)\,(\forall y)\,[\sim P(x) \vee \sim S(y) \vee \sim H(x, y)]$

Conversion to the Normal Form

6. Convert the matrix to conjunctive normal form using: $p \vee (q \wedge r) = (p \vee q) \wedge (p \vee r)$
   - ❑ Our example is already in conjunctive normal form.
7. Drop the universal quantifiers as unnecessary.
   - 1) $\Rightarrow$ $P(a) \wedge (\sim F(y) \vee H(a, y))$
   - 2) $\Rightarrow$ $\sim P(x) \vee \sim S(y) \vee \sim H(x, y)$
   - 3) $F(b) \wedge S(b)$

Conversion to the Normal Form

8. Eliminate $\wedge$ signs by writing the wff as a set of clauses.
   - 1) $\Rightarrow$ $\{\, P(a), \sim F(y) \vee H(a, y) \,\}$
   - 2) $\Rightarrow$ $\{\, \sim P(x) \vee \sim S(y) \vee \sim H(x, y) \,\}$
   - 3) $\Rightarrow$ $\{\, F(b), S(b) \,\}$

   Or:

1a)	$P(a)$
1b)	$\sim F(y) \vee H(a, y)$
2a)	$\sim P(x) \vee \sim S(y) \vee \sim H(x, y)$
3a)	$F(b)$
3b)	$S(b)$

Conversion to the Normal Form

9. Rename variables in clauses so that each clause has unique variable name.

1a)	$P(a)$
1b)	$\sim F(y) \vee H(a, y)$
2a)	$\sim P(x) \vee \sim S(z) \vee \sim H(x, z)$
3a)	$F(b)$
3b)	$S(b)$

Unification

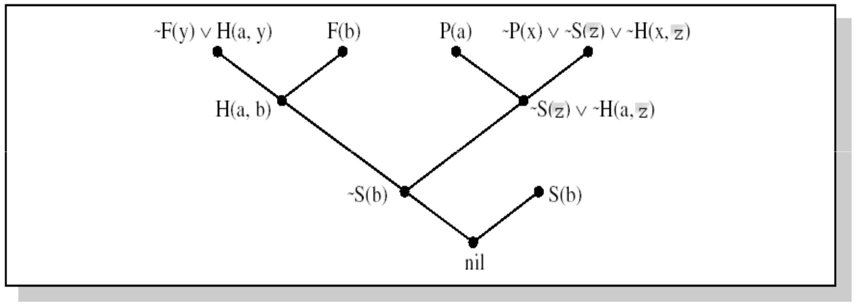
- The process of finding substitutions for variables to make arguments match is called unification.
- Without unification it is impossible to resolve clauses such as:

  $\sim F(y) \vee H(a, y)$

  $F(b)$
- For (1a) – (3b): $\{x/a, y/b, z/b\}$

Proof by Resolution Refutation

Figure 3.20 Resolution Refutation Tree to Prove No Failure Is a Success



$\therefore$ $P(a) \wedge (\sim F(b) \vee H(a, b)) \wedge$
$(\sim P(a) \vee \sim S(b) \vee \sim H(a, b)) \wedge F(b) \wedge S(b) = nil$

- Since the root is nil, the conclusion is valid.

Chaining

- Chain – a group of multiple inferences that connect a problem with its solution
- A chain that is searched / traversed from a problem to its solution is called a forward chain.
- A chain traversed from a hypothesis back to the facts that support the hypothesis is a backward chain.
- Problem with backward chaining is to find a chain linking the evidence to the hypothesis.

Figure 3.21 Causal Forward Chaining

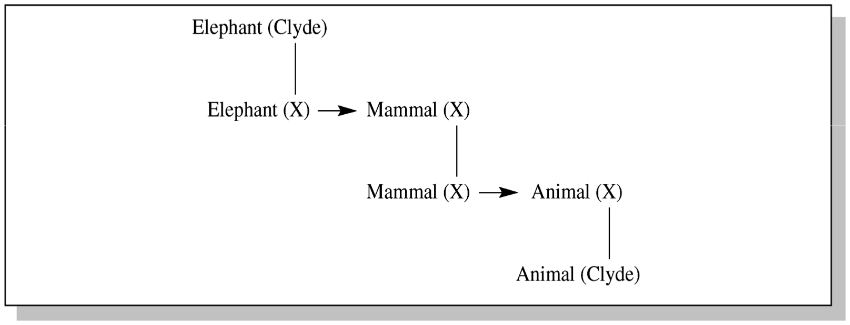
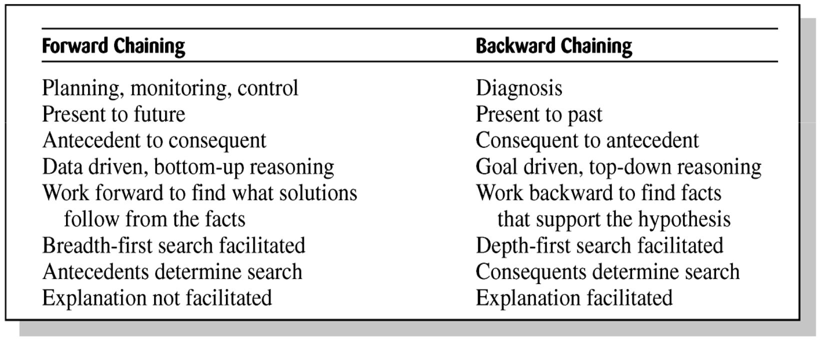


Table 3.14 Some Characteristics of Forward and Backward Chaining



Forward Chaining	Backward Chaining
Planning, monitoring, control	Diagnosis
Present to future	Present to past
Antecedent to consequent	Consequent to antecedent
Data driven, bottom-up reasoning	Goal driven, top-down reasoning
Work forward to find what solutions follow from the facts	Work backward to find facts that support the hypothesis
Breadth-first search facilitated	Depth-first search facilitated
Antecedents determine search	Consequents determine search
Explanation not facilitated	Explanation facilitated

Figure 2.7 Depth-First and Breadth-First Searches for an Arbitrary Tree

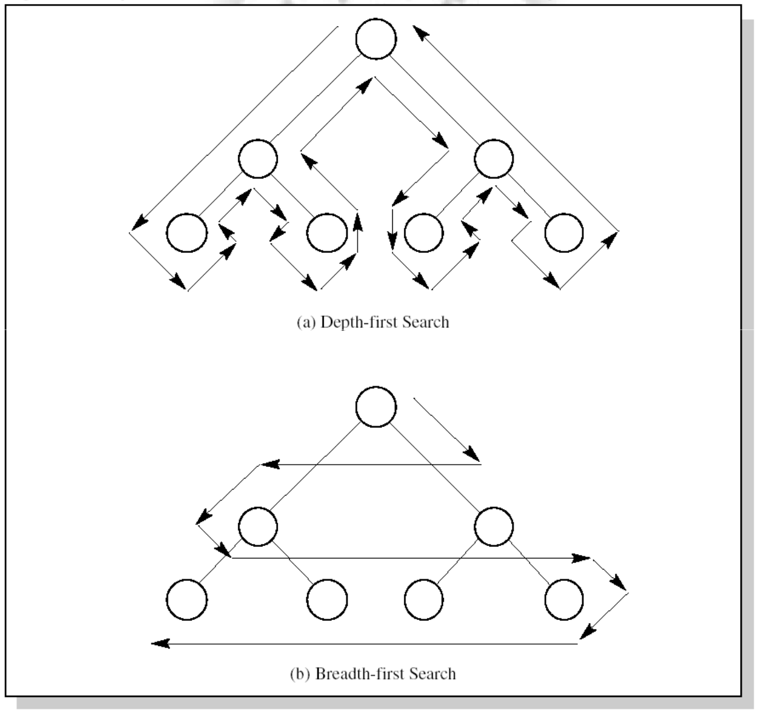
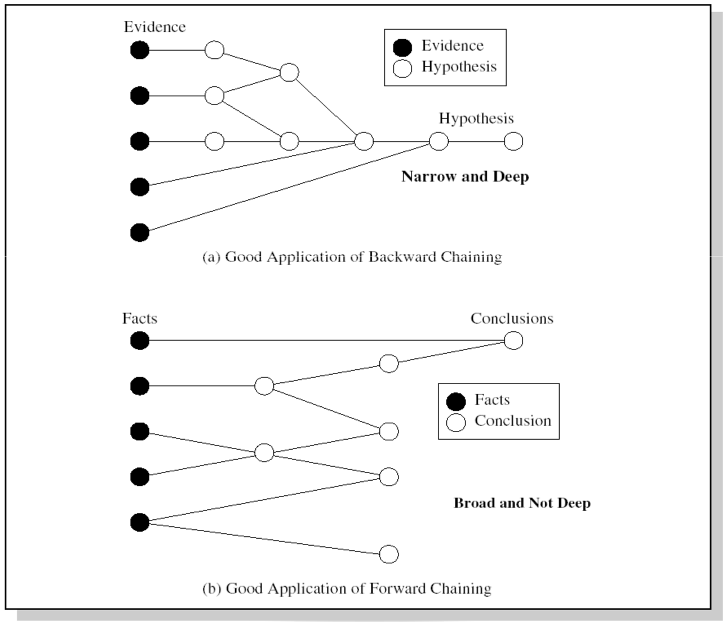


Fig. 3.25: Forward and Backward Chaining



Other Inference Methods

- Analogy – relating old situations (as a guide) to new ones.
  - Not a formal proof but a heuristic reasoning
  - May or may not work
  - Primary used in medical diagnosis and legal arguments
- Generate-and-Test – generation of a likely solution then test to see if proposed meets all requirements.
- Abduction – Fallacy of the Converse
  - Used by some diagnostic systems to guess the disorder from the symptoms and then test it (generate-and-test)

Other Inference Methods

- Deductive logic is monotonic system
  - Addition of new axioms may increase new theorems.
  - New axioms can not contradict the old ones.
- Nonmonotonic Reasoning – theorems may not increase as the number of axioms increase.
  - New axioms can contradict the old ones.
  - If new axioms contradict old ones, the theorems proved from the old ones will be removed.
- In practice truth maintenance of the rules is difficult and the nonmonotonicity is accomplished by adding special operators to avoid contradiction.

Other Inference Methods

- The following example prevents the incorrect rule from firing, instead of retracting the invalid conclusion:

IF X is a bird THEN X can fly.

<u>Tweety is a bird.</u>

∴ Tweety can fly. (What if Tweety is a penguin ?)

⇓

IF x is a bird AND x is typical THEN x can fly.

IF x is a bird AND x is not typical THEN x cannot fly.

Tweety is a bird. ⇒ If Tweety is a penguin, just add:

<u>Tweety is nontypical.</u>

∴ Tweety cannot fly.

Metaknowledge

- Knowledge about a pre-selected knowledge.
  - E.g. Path planning for robot
- The Markov decision process (MDP) is a good application to path planning for a robot.
- In the real world, there is always uncertainty, and pure logic is not a good guide when there is uncertainty.
- A MDP is more realistic in the cases where there is partial or hidden information about the state and parameters, and the need for planning.

Metaknowledge

- MDF can be defined as a tuple:
  - MDF ≡ {states, actions, transitions, rewards}
    - State: a set of states of the environment
    - Action: valid operations
    - Transition: determines what the next state will be for a particular action
- The goal is to maximize the sum of rewards in the search for the optimum path to the goal.
- Can determine the hidden parameters from the observable parameters.
  - Robot is not sure of its location – an unknown state, or which path to take – an unknown action

Summary

- We have discussed the commonly used methods for inference for expert systems.
- Expert systems use inference to solve problems.
- We discussed applications of trees, graphs, and lattices for representing knowledge.
- Deductive logic including syllogistic, propositional, and first-order predicate logic were discussed.
- Venn diagrams and truth tables were discussed as means of proving theorems and statements.

Summary

- Characteristics of logic systems were discussed.
- Resolution as a means of proving theorems in propositional and first-order predicate logic.
- The nine steps to convert a wff to clausal form were covered.