

Artificial Intelligence

Unit-V

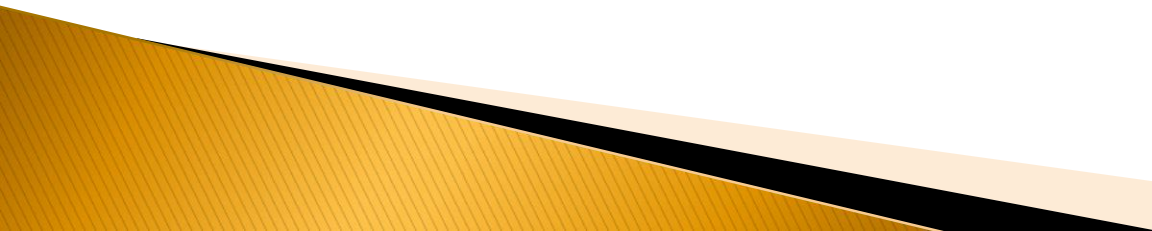
Expert System and Applications

V Semester CSE


Course Outcomes

After successful completion of this course, the student will be able to:


CO	Course Outcomes	Knowledge Level
1	Illustrate the concept of Intelligent systems and current trends in AI	K2
2	Apply Problem solving, Problem reduction and Game Playing techniques in AI	K3
3	Illustrate the Logic concepts in AI	K2
4	Explain the Knowledge Representation techniques in AI	K2
5	Describe Expert Systems and their applications	K2
6	Illustrate Uncertainty Measures	K2

- Introduction
 - Phases in building Expert Systems
 - Expert Systems vs Traditional Systems
 - Rule-based Expert Systems
 - Blackboard Systems
 - Truth Maintenance Systems
 - Applications of Expert Systems
- 


Introduction

- One of the goals of AI is to understand the concept of intelligence and develop intelligent computer programs. An example of a computer program that exhibits intelligent behavior is an Expert System
 - Expert Systems are meant to solve real-world problems which require specialized human expertise and provide expert quality advice, diagnoses, and recommendations
 - An ES is a software program or system that tries to perform tasks similar to human experts in a specific domain of the problem
 - Expert systems are also called Knowledge-based expert systems
- 


Introduction

- The term expert systems is often reserved for programs whose knowledge base contains the knowledge provided by human experts in contrast to knowledge gathered from textbooks or non-experts.
 - For eg: MYCIN was developed using the expertise of best diagnosticians of bacterial infections
 - In a chemical refinery, a knowledge engineer was assigned to produce an ES to reproduce the expertise of an experienced retired employee to save the company from incurring loss
 - The power of an ES lies in its store of knowledge regarding the problem domain; the more knowledge a system is provided, the more competent it becomes.
- 


Phases in Building Expert Systems

- **Identification Phase:** In this phase, the knowledge engineer determines important features of the problem with the help of the human domain expert. The parameters that are determined in this phase are the type and scope of the problem, the kind of resources required, and the goal and objective of the ES
 - **Conceptualization Phase:** In this phase, the knowledge engineer and domain expert decide the concepts, relations, and control mechanism needed to describe the problem-solving method. Issue of granularity is also addressed
- 

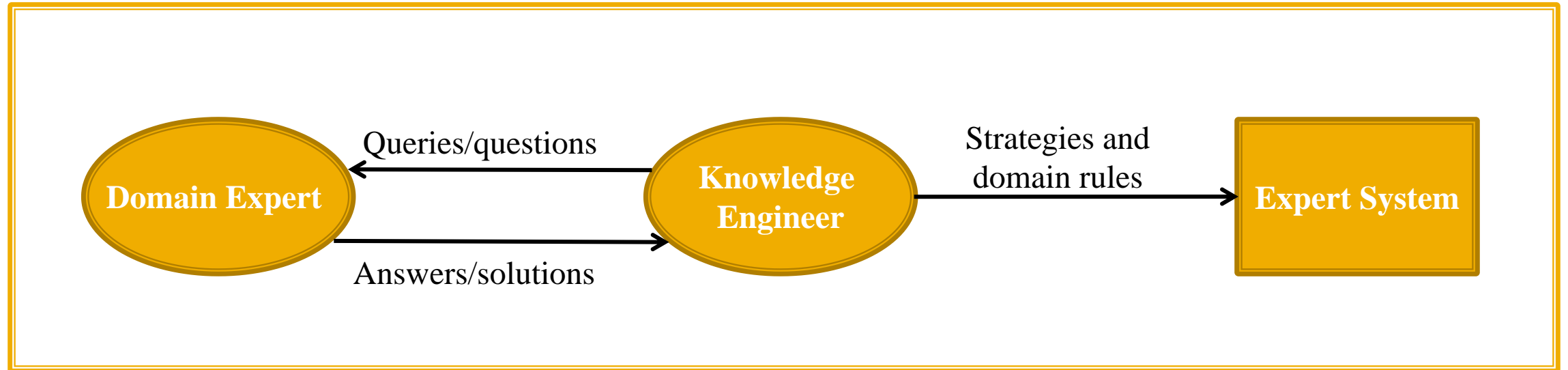
Phases in Building Expert Systems

- **Formalization Phase:** This phase involves expressing the key concepts and relations in some framework supported by ES building tools. Formalized knowledge consists of data structures, inference rules, control strategies and languages required for implementation
 - **Implementation Phase:** During this phase, formalized knowledge is converted to a working computer program, initially called prototype of the whole system
 - **Testing Phase:** This phase involves evaluating the performance and utility of prototype system and revising the system, if required. The domain expert evaluates the prototype system and provides feedback, which helps the knowledge engineer to revise it
- 

Knowledge Engineering


- The whole process of building an ES is called Knowledge Engineering
 - It involves interaction between ES builder, or the knowledge engineer, one or more domain experts, and potential users
 - The tasks and responsibilities of knowledge engineer are:
 - Ensuring that the computer has all the knowledge needed to solve a problem
 - Choosing one or more forms to represent the required knowledge
 - Ensuring that the computer can use the knowledge efficiently by selecting some of the reasoning methods
- 

Knowledge Engineering




Interaction between Knowledge Engineer and Domain Expert for creating an ES


Knowledge Engineering

- The job of the knowledge engineer involves close collaboration with the domain experts and end users.
 - The knowledge engineer become familiar with the problem domain by reading texts or literature and talking to the domain experts and interviewing them.
 - He / She then extract general rules from the discussion and interview held with expert(s) and get them checked by the expert(s) for correctness.
 - The engineer then translates the knowledge into a computer usable language and designs an inference engine, which is a reasoning structure that uses the knowledge appropriately.
- 

Knowledge Engineering

- The domain knowledge, consisting of both forms, textbook knowledge and experiential knowledge, is entered into the program piece by piece
 - The basic development cycle should include the development of an initial prototype and iterative testing and modification of that prototype by both experts and users
 - In order to develop the initial prototype, the knowledge engineer will have to take decisions regarding appropriate knowledge representation(e.g., rules, semantic net, or frames) and inference methods(e.g., forward chaining or backward chaining or both)
 - To test these basic design decisions, the first prototype may be so designed that it only solves a small part of the overall problem
- 


Knowledge Representation

- The most important ingredient in any ES is knowledge
 - The power of ES resides in the specific, high-quality knowledge they contain regarding problem domains
 - The collection of domain knowledge is called knowledge base, while the general problem-solving knowledge may be called inference engine, user interface, etc.
 - The most common knowledge representation scheme for ES consists of production rules; they are of the if-then form, where the if part contains a set of conditions and then part contains conclusion
 - If the ‘**if**’ part of the rule is satisfied then the ‘**then**’ part can be concluded.
 - ES in which knowledge is represented in the form of rules are called rule-based systems.
 - Another widely used representation in ES is called the unit (also known as frame, semantic network, etc.), which is based upon more passive view of knowledge.
- 

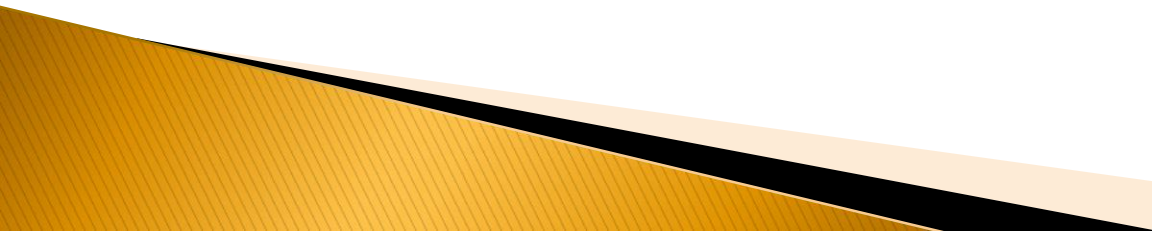
Expert Systems versus Traditional Systems

Traditional Systems	Expert Systems
It manipulates data	It manipulates knowledge
Problem expertise is encoded in program as well as in the form of data structures	Problem expertise is encoded in data structures only
It performs tasks using conventional decision making logic	Small fragments of human experience are collected into a knowledge base to solve problem
If the knowledge changes, the program has to be rebuilt	A different problem, within the domain of the knowledge base, can be solved using the same program without having to reprogram the system
They don't use confidence factor or certainty factors.	They allow the use of confidence or certainty factors. They imitate the confidence humans use in reasoning. For eg. "If weather is humid, then it might probably rain." There is a lot of uncertainty in the statement.
Programs are designed always to produce correct results.	They are designed to behave like human experts and may some times produce incorrect results.


Characteristics of Expert Systems

- **Expertise:** An ES should exhibit expert performance, have high level of skill, and possess adequate robustness
 - **Symbolic reasoning:** Knowledge in an ES is represented symbolically which can be easily reformulated and reasoned
 - **Self knowledge:** A system should be able to explain and examine its own reasoning
 - **Learning capability:** A system should learn from its mistakes and mature as it grows
 - **Ability to provide training:** Every ES should be capable of providing training by explaining the reasoning process behind solving a particular problem using relevant knowledge
 - **Predictive modelling power:** The system can act as an information processing model of problem solving. It can explain how new situation led to the change, which helps users to evaluate the effect of new facts and understand their relationship to the solution
- 

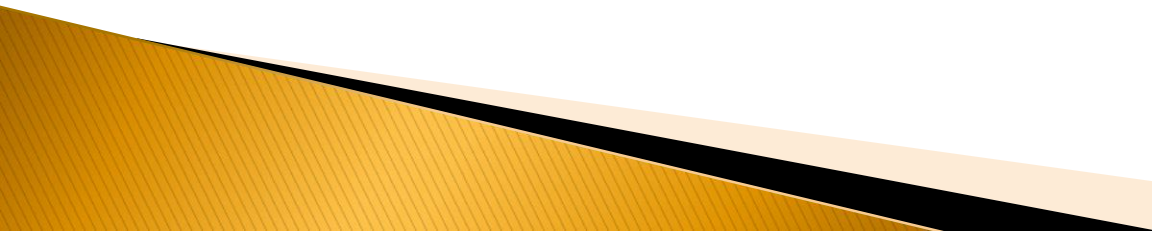
Evaluation of Expert Systems

- Evaluation of an ES consists of **Performance and Utility Evaluation**.
 - **Performance evaluation** consists of answering various questions such as given below:
 - Does the system make decisions that experts generally agree to?
 - Are the inference rules correct and complete?
 - Does the control strategy allow the system consider items in the natural order that the expert prefers?
 - Are relevant questions asked to the user in proper order?
 - Is the explanation given by the ES adequate for describing **how** and **why** conclusions?
- 


Evaluation of Expert Systems

- **Utility evaluation** consists of answering the following questions:
 - Does the system help user in some significant way?
 - Are the conclusions of the system organized and ordered in a meaningful way?
 - Is the system fast enough to satisfy the user?
 - Is the user interface friendly enough?
- 


Justification of the need for developing an Expert System

- One should evaluate whether a problem is suitable for an ES solution using the following guidelines:
 - Specialized Knowledge problems –For eg. Oil exploration and medicine
 - High Payoff
 - Existence of cooperative experts: Experts should be willing to help and provide their expertise
 - Justification of cost involved in developing ES: Realistic assessment of cost and benefits involved
 - The type of problem : Problem must be structured and should not require common sense knowledge
- 


Advantages of Expert Systems

- Helps in preserving scarce expertise
 - Provides consistent answers for repetitive decisions, processes and tasks
 - Fastens the pace of human professional or semi-professional work
 - Holds and maintains significant levels of information
 - Provides improved quality of decision making
 - Domain experts are not always able to explain their logic and reasoning unlike ES
 - Encourages organizations to clarify the logic of their decision-making
 - Leads to major internal cost savings within companies
 - Causes introduction of new products
 - Never forgets to ask a question, unlike a human
- 

Disadvantages of Expert Systems

- Unable to make creative responses as human experts would in unusual circumstances
 - Lacks common sense needed in some decision making
 - May cause errors in the knowledge base, and lead to wrong decisions
 - Cannot adapt to changing environments, unless knowledge base is changed
- 

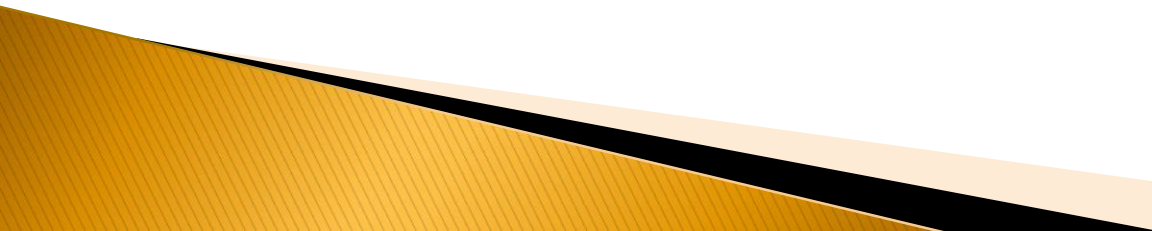
Languages for ES Development

- The basic hypothesis of AI is that intelligent behavior can be described as symbol manipulation and can be modelled with the symbol processing capabilities of computer
 - One of the most prominent languages suitable for AI programming is LISP (LISt Processing), which is based on **lambda calculus**. It is a simple, elegant, and flexible language; most AI reaserch programs are written in LISP.
 - Another AI programming language, known as Prolog(PROgramming LOGic). It is based on first-order predicate calculus. Prolog programs behave in a manner similar to rule-based systems.
 - Nowadays, Object oriented languages (C++, Java etc.) and even C are used for developing ES
- 

Rule-based Expert Systems

- Rule-based systems can be either goal driven or data driven
- Expert systems may use either one or both strategies, but the most common is the goal-driven/backward-chaining strategy

Expert System Shell in Prolog

- A simple expert system shell which uses backward chaining control strategy is considered
 - Prolog uses Backward Chaining
 - We define a special syntax for the rules using operator declarations(`:-op`). Using `op`, any standard system operator declaration can be changed or new operator can be defined by the user. A goal `op` with three arguments is shown next:
- 

Expert System Shell in Prolog

- `:-op(Prec , Type, Atom)`
- Where Prec is an unassigned integer in the range [1,1200], $Type \in \{fx, fy, xfx, xfy, yfx, yfy\}$ is a specification representing associativity, and Atom is a symbol or a name declared as an operator with precedence Prec and Type.
- For declaring infix operators, possible values of specification Type are xfx, xfy, yfx, and yfy where f represents operator and x and y represents arguments

`:- op(500,yfx, +).` ---indicates left associative

`:- op(500,yfx, -).`

`:- op(300,xfx, mod).`

`:- op(700,xfx, >)` ---- indicates no associativity.

- The character x means the operator in the argument must have strictly lower precedence value than the operator f and y means the argument can contain operator with the same or lower precedence value than the operator f
- yfx is left associative and xfy is right associative

Expert System Shell in Prolog

- The Type fx indicates non-associative(prefix) operators, fy denotes right-associative(prefix) operators, xf implies non-associative(postfix) operators, and yf is used for left-associative(postfix) operators. For example, logical negation(\sim) is defined as:

`:- op(900,fy, ~)`

- Similarly, op allows us to define operators if and then as non-associative prefix operator and right-associative infix operator, respectively
- We can handle expressions of the form: `if[symptom(S1), symptom(S2)] then disease (D).`

Expert System Shell in Prolog

Operator Declaration

`:- op(975, fx, if)`

`:- op(950, xfy, then).`

`Consult('knowledge_base') %load knowledge base in the working expert`

Knowledge extraction module from Expert

% Get the name of diseases([flu, cold, measles, chicken_pox, mumps]) from the expert

`expert_input` `:-` `writeln('Input the list of diseases as [d1, d2, d3...]'); readln(D),`
 `asserta(diseases(D), 'knowledge_base'), !, input_sym(L),`
 `input_disease(C), asserta(rule(if L then C), 'knowledge_base').`

`input_sym(L)` `:-` `writeln('Input the list of symptoms as [s1, s2, s3, ...]'), readln(L).`

`input_diseases(L)` `:-` `writeln('Input the corresponding Disease'), readln(L).`

Expert System Shell in Prolog

Knowledge Base generated by knowledge extraction module

```
diseases([measles, flu, cold, mumps, cough, chicken_pox]).  
rule(if [fever, cough, running_nose, conjunctivitis, rash]) then measles).  
rule(if [fever, headache, body_ache, sore_throat, cough, chills, running_nose, conjunctivitis] then flu).  
rule(if [headache, sneezing, chills, sore_throat, running_nose] then cold).  
rule(if [fever, swollen_glands] then mumps).  
rule(if [cough, sneezing, running_nose] then cough).  
rule(if [fever, chills, body_ache, rash] then chicken_pox).
```


Expert System Shell in Prolog

User Query Module

```
get_symptom(X)      :-      user_response(['Do you have', X, '? Type(y/n)'], X).  
% if the symptom has already been entered earlier then no need to enter again  
user_response(_, X)  :-      yes(X), !.  
% if symptom is the new one then assert it in working memory appropriately  
user_response(Q, X)  :-      ask_query(Q, X, R), R = 'y'.  
ask_query(Q, X, R)   :-      writeln(Q), readln(R), store(X, R).  
store(X, 'y')        :-      asserta(yes(x)).  
store(X, 'n')        :-      asserta(no(X)).
```

Expert System Shell in Prolog

Main ES Shell

consultation	:-	writeln ('Welcome to diagnostic System'), writeln('Input your name), readln(Name), hypothesis(D).
hypothesis(D)	:-	diseases(D), member(G, D), check(G), !, writeln(Name, 'probably has', G), clear_consult_facts.
hypothesis(D)	:-	writeln('Sorry, not able to diagnose'), clear_consult_facts.
check(G)	:-	rule(if L then G), check_symptom(L).
check_symptom([]).		
check_sumptom([G Gs])	:-	get_symptom(G),!, check_symptom(G),!check_symptom(Gs).
check_symptom(G)	:-	no(G), fail.
check_symptom(G)	:-	yes(G)
clear_consult_facts	:-	retractall(yes(_)).
clear_consult_facts	:-	retractall(no(_)).

Problem-independent Forward Chaining

Forward inference mechanism is a reasoning process that begins with known facts and proceeds forward to find a successful goal. Although Prolog uses backward chaining as a control strategy, it is possible to write a program in Prolog which uses forward chaining concept.

Prolog Program	Corresponding Facts
a :- b , c.	rule(if[b, c] then a).
c :- b , e, f.	rule(if[b, e, f] then c).
b.	fact(b)
e.	fact(e)
f.	fact(f)

Here a, b, c, e and f represents atoms(predicates with arguments, if any). Let us store newly created facts(rule and fact) in a database file rule_facts_file, which is consulted at the time of executing the forward-chaining program. The program is shown in the next slide.

Representation of Rules as Facts

Forward Chaining

```
:- op(975, fx, if).  
:- op(950, xfy, then).  
consult('rule_facts_file').
```

% forward predicate continues till all facts are exhausted. For each fact, it tries all the %rule_facts having fact in the body of the rule. Delete this fact and put it in used_fact list

```
forward                :-    not(fact(X)), !.                                (1)
```

```
forward                :-    fact(X), not(try_all(rule_fact(X)) ),  
                             assertz(used_facts(X)), retract(fact(X)), forward.    (2)
```

```
try_all(P)             :-    call(P), fail.                                (3)
```

% For each rule if fact is in the body then either new rule or new fact is generated depending upon whether the body of the rule becomes empty after deleting the fact from the body

```
rule_fact(X)           :-    rule(if Body then Head), rule1(X, Body, Head).        (4)
```

```
rule1(X, B, H)         :-    member(X, B), delete(X, B, NBody), rule2(NBody, H).    (5)
```

% Assert the rule in working memory or the fact and display it also for the user

```
rule2([], Head)        :-    not(fact(Head)), asserta(fact(Head)), writeln('Goal proved is ', Head).    (6)
```

```
rule2(Body, Head)      :-    not(Body = []), asserta(rule(if Body then Head)).    (7)
```


A Query Trace

Rule applied	Bindings	Trace	Comments
	Main Goal	?- forward	
2		?- fact(X), not(try_all(rule_fact(X))), assertz(used_facts(X)), retract(fact(X)), forward.	It fails with rule (1)
	{X = b}	?- not(try_all(rule_fact(X))), assertz(used_facts(X)), retract(fact(X)), forward.	
3		?- rule_fact(b),	
4	{Body = [b, c], Head = a}	?- rule(if Body then Head), rule1(b, Body, Head),	
5		?- rule1(b, [b, c], a), ...	
	{NBody = [c], H = a}	?- member(b, [b, c]), delete(b, [b,c], NBody), rule2(NBody, H),	
7		?- rule2([c], a), ... ?- not([c] = []), asserta(rule(if [c] then a)), ... then a)	Assert rule([if [c] then a])
	{Body=[b,e, f], Head = c}	?- rule(if Body then Head), rule1(b, Body, Head),	Backtrack

A Query Trace

Rule applied	Bindings	Trace	Comments
5	{NBody= [e, f], H = c}	?- rule1(b, [b, e, f], c), ... ?- member(b, [b, e, f], delete(b, [b, e, f], NBody), rule2(NBody, H),	
7		?- rule2([c], c), ... ?- not([e, f] = []), asserta(rule(if [e, f] then c)), ...	Assert Rule([if [e, f] then c])
7		?-assertz(used_facts(b)), retract(fact(b)), forward.	Backtrack, Assert used_fact(b) and delete fact(b)
2	{X = e}	?- forward. ?- not(try_all(rule_fact(X))), assertz(used_facts(X)), retract(fact(X)), forward.	
3		?- rule_fact(e), Continue like this. We get the goals c and then a displayed	


Status of Rule Facts File during Execution of Forward Chaining Predicate

Facts	Rules	Comments	Display
fact(b). fact(e). fact(f).	rule(if [b, c] then a). rule (if [b, e, f] then c).	Initial rule_fact_file	
fact(e). fact(f). used_facts(b).	rule (if [e, f] then c) rule(if [c] then a). rule(if [b, c] then a). rule (if [b, e, f] then c).	rule_fact_file status after fact 'b' tried all the rules	
fact(f). used_facts(b). used_facts(e).	rule (if [b, f] then c). rule (if [f] then c) rule (if [e, f] then c) rule(if [c] then a). rule(if [b, c] then a). rule (if [b, e, f] then c).	rule_fact_file status after fact 'e' tried all the rules	
fact(c). used_facts(b). used_facts(e). used_facts(f).	rule (if [b] then c). rule (if [b, f] then c). rule (if [f] then c) rule (if [e, f] then c) rule(if [c] then a). rule(if [b, c] then a). rule (if [b, e, f] then c).	rule_fact_file status after fact 'f' tried all the rules	fact(c) is asserted in this file. 'c' is displayed on screen


Status of Rule Facts File during Execution of Forward Chaining Predicate

Facts	Rules	Comments	Display
fact(a). used_facts(b). used_facts(e). used_facts(f). used_facts(c)	rule (if [b] then c). rule (if [b, f] then c). rule (if [f] then c) rule (if [e, f] then c) rule(if [c] then a). rule(if [b, c] then a). rule (if [b, e, f] then c).	rule_fact_file status after fact 'c' tried all the rules	fact(a) is asserted in this file. 'a' is displayed on screen
used_facts(b). used_facts(e). used_facts(f). used_facts(c) used_facts(a)	rule (if [b] then c). rule (if [b, f] then c). rule (if [f] then c) rule (if [e, f] then c) rule(if [c] then a). rule(if [b, c] then a). rule (if [b, e, f] then c).	rule_fact_file status after fact 'a' tried all the rules	Since no fact left, forward predicate terminates

ES Shells and Tools

- There are only a few AI methods, such as knowledge representation, inferences strategies, etc., essentially required in building expert systems, but there is a wide variation in domain knowledge. We can develop systems containing useful methods without any domain-specific knowledge. Such systems are known as skeletal systems, shells, or simple AI tools. Most expert systems are developed using these specialized software tools.
 - A shell is a complete development environment that may be used for building and maintaining knowledge-based applications. It provides knowledge engineer with a step-by-step methodology for the domain experts to be directly involved in structuring and encoding the knowledge. These shells have in-built inference mechanism(backward chaining, forward chaining, or both) and required knowledge to be entered according to a specified format while developing ES.
 - Using shells for building expert systems has some advantages such as reduction in development time and costs. A large number of commercial shells are available for all types of systems such as PCs, workstations and large mainframe computers. An important point regarding shells is that although they simplify programming, they do not help with knowledge acquisition.
- 

MYCIN Expert System and Various Shells

- MYCIN is one of the oldest expert systems. The design of most of the commercial expert systems and shells has been influenced by that of MYCIN. It was developed as an ES that could diagnose and recommend treatment for certain blood infections. MYCIN was developed for exploring the ways in which human experts make guesses on the basis of partial information.
 - In MYCIN, the knowledge is represented as a set of if-then rules with certainty factors.
 - The MYCIN rule may be written in English for the sake of clarity in the following manner:
 - **IF the infection is primary-bacteremia AND the site of the culture is one of the sterile sites AND the suspected portal of entry is the gastrointestinal tract. THEN there is suggestive evidence(0.7) that infection is bacteroid**
 - Interpretation of the rule described above is that the conclusion bacteroid will be true with a certainty factor 0.7 given all the evidences are present
- 


MYCIN Expert System and Various Shells

A user conversation with MYCIN had three stages:

1. In the first stage, initial data regarding the case was gathered to enable the system to come up with a very broad diagnosis
2. In the second stage, the questions that were asked to test specific hypotheses were more direct in nature
3. In the final stage, a diagnosis was proposed

More questions were asked to determine an appropriate treatment and given the diagnosis and facts about the patient, a treatment was recommended. At any stage, the user could put up queries regarding why a particular question was asked or how a conclusion was reached

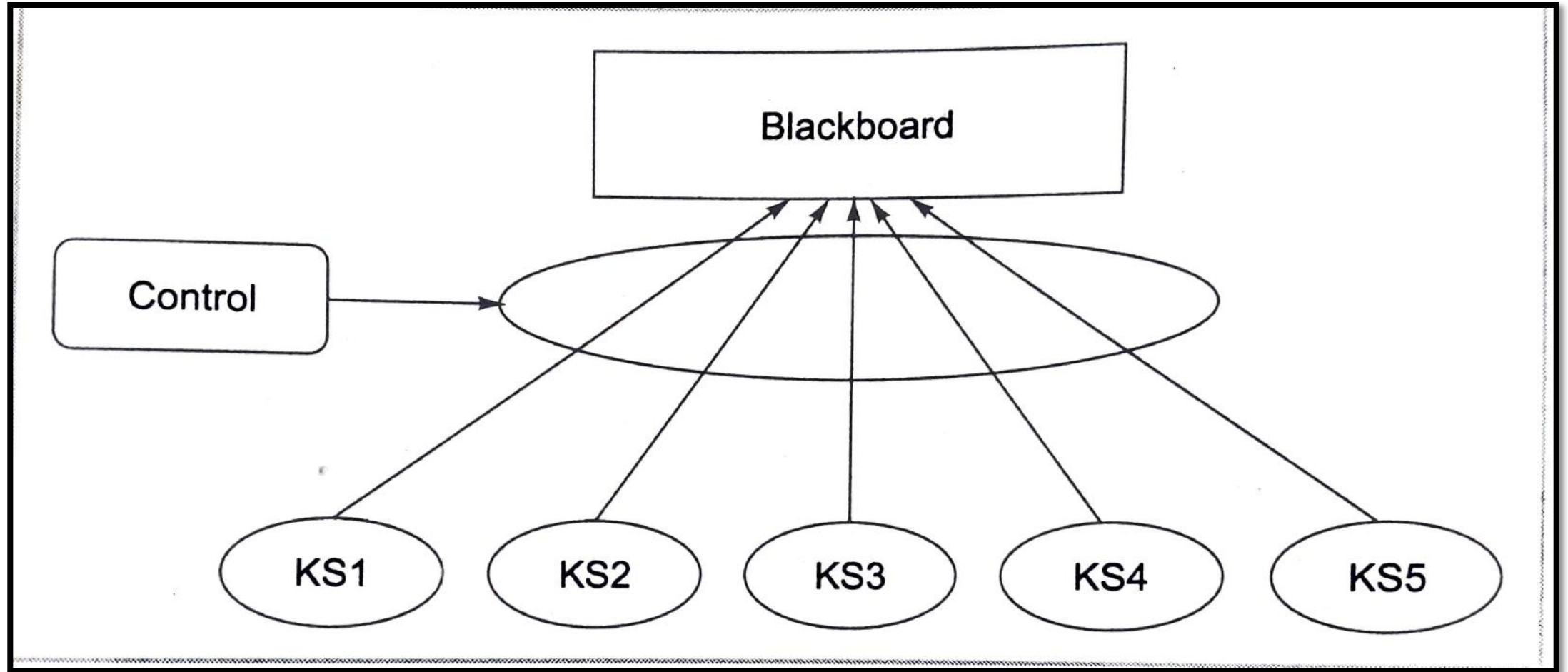
The basic problem-solving strategy involved going down the disease tree, from general classes of diseases to very specific ones, gathering information to differentiate between two disease subclasses. For example, if disease1 has subclasses disease2 and disease3, and the system has established that a patient has disease1, and subclasses disease2 and disease3 are differentiated by only a few symptoms, then MYCIN would ask about differentiating symptoms to confirm



Blackboard Systems

- In blackboard systems, indirect and anonymous communication approach among modules with the help of an intermediary, such as a blackboard data repository, proves to be extremely useful. In this approach, all processing paths are possible, and a separate moderator mechanism dynamically selects a path among the possible paths. The information placed on the blackboard is public and is made available to all modules
- Blackboard systems were developed to solve complex, difficult, and ill-structured problems in a wide range of application areas. Blackboard architecture is a way of representing and controlling the knowledge bases; using independent group of rules called Knowledge Sources(KSs) that communicate through data control database called a blackboard

Blackboard Systems

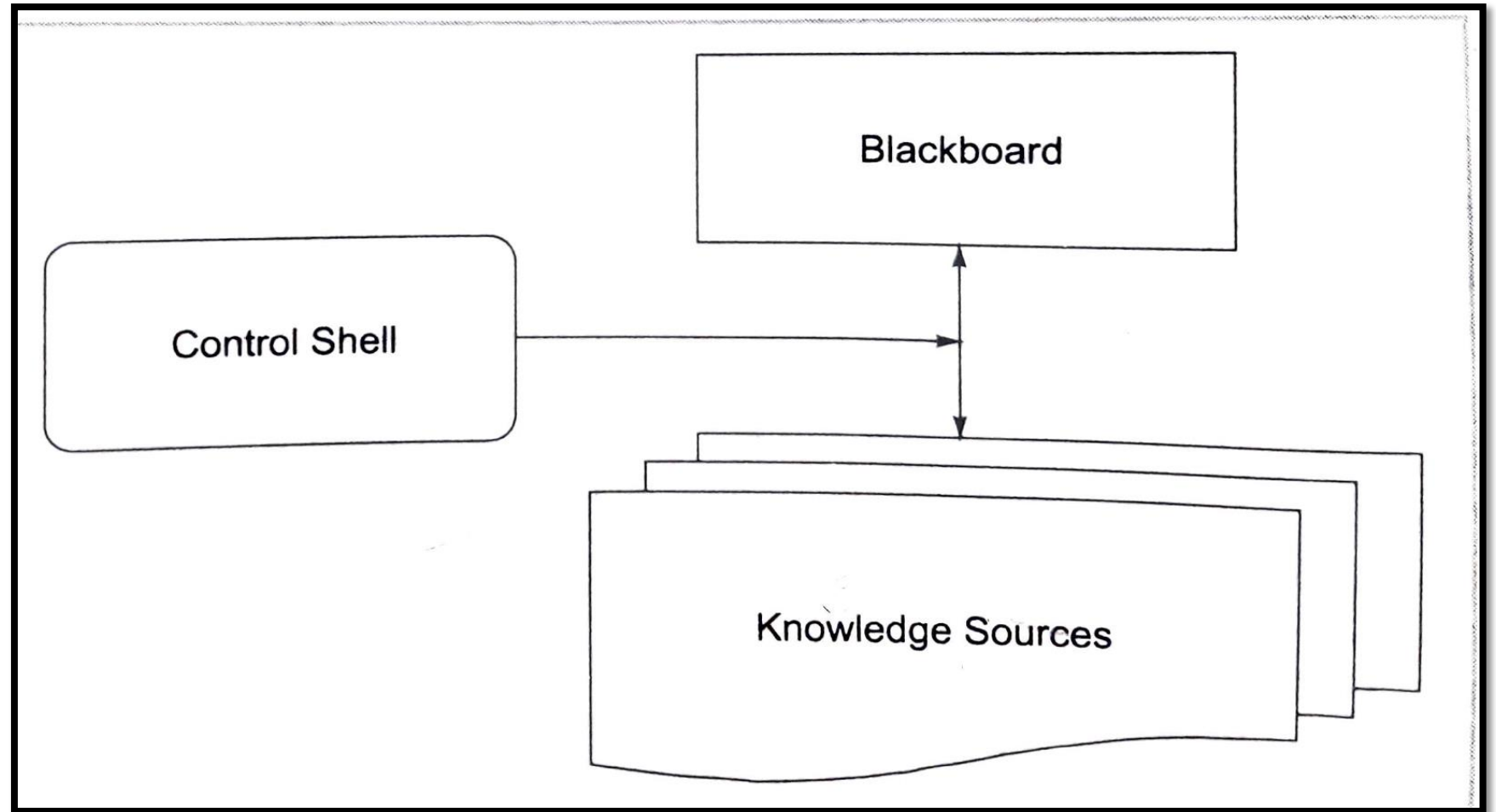


Indirect and Anonymous Communications

Blackboard Systems

The main modules of a blackboard system:

- i. Knowledge sources
- ii. Blackboard
- iii. Control shell




Blackboard System

Blackboard Systems

i. Knowledge sources

Blackboard systems use a functional modularization of expertise knowledge in the form of Knowledge Sources(KSs). These are independent computational modules that contain the expert knowledge needed to solve a given problem. A KS is regarded to a specialist at solving certain aspects of the overall application and is separate and independent of all other KSs in the blackboard system. It is possible to add additional KSs to the blackboard system and upgrade or even remove existing KSs. Each KS is aware of the conditions under which it can contribute toward solving a particular problem. This knowledge in problem-solving process is known as *triggering condition*



Blackboard Systems

ii. Blackboard

Blackboard represents a global data repository and shared data structure available to all KSs. It contains raw input data, partial solutions, alternatives and final solutions, control information, communication medium and buffer, and a KS trigger mechanism used in various phases in problem-solving. A blackboard is subdivided into regions, with each region corresponding to a particular kind of information.

An important characteristic of the blackboard approach is its ability to integrate contributions dynamically. An advantage of the blackboard system is that the system can retain the results of problems that have been solved earlier, thus avoiding the task of re-computing them later.




Blackboard Systems

iii. Control component

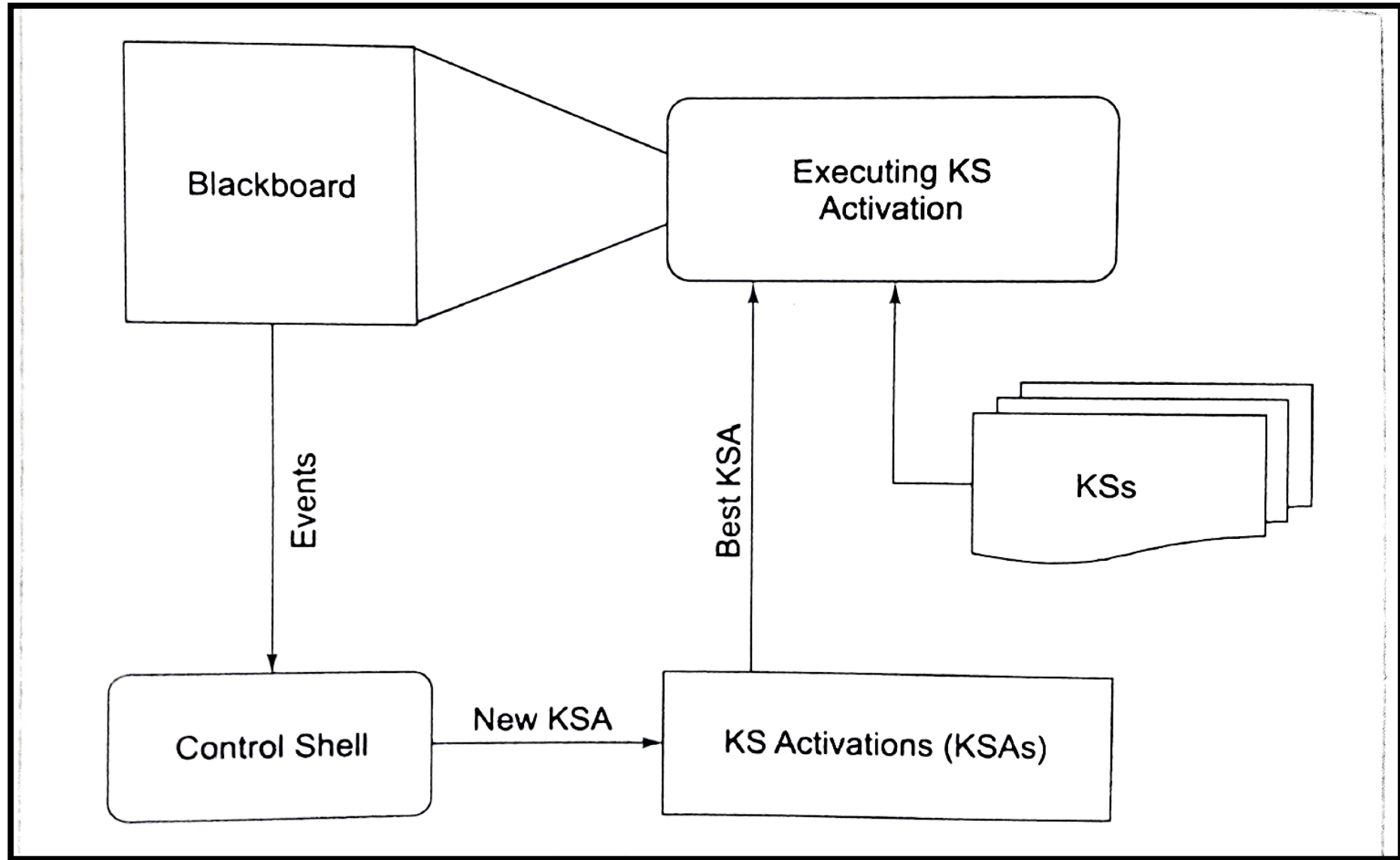
A control component of a blackboard system helps in making runtime decisions regarding the course of problem solving and the expenditure of problem solving resources. Control component is separate from the individual KSs.

Sometimes a control shell directs the problem-solving process by allowing KSs to respond opportunistically to changes made to the blackboard. A blackboard system uses the process of incremental reasoning, i.e. the solution is built one step at a time.

In a classic blackboard-system control approach, the currently executing KS activation(KSA) generates events as it makes changes on the blackboard. These events are ranked and maintained until the executing KSA is completed. The control shell uses these events to trigger and activate KSs. Out of all the ranked KSAs, the best KSA is selected for execution



Blackboard Systems

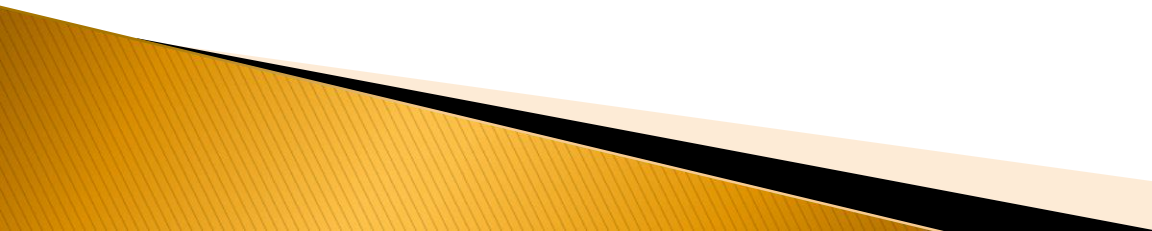


Blackboard-System Control Cycle

Blackboard Systems

Knowledge Source Execution Method

Steps involved in most of KS executions in a blackboard system are:

- The control shell is notified of an event of interest to the KS
 - This triggering information is used by the control shell to activate the KS in case such events occur
 - KS uses the triggering context to determine the ranges of attribute values and searches the blackboard to find blackboard objects possessing attributes within those ranges
 - The retrieved objects triggering context information are used by the KS to perform its computations
 - The results of this computation are written onto the blackboard
- 

Blackboard Systems

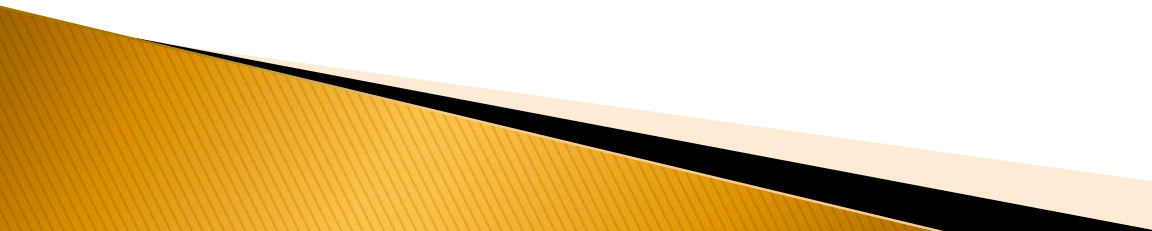
Issues in Blackboard Systems for Problem Solving

Information can be represented on blackboard in two ways:

- Specialized Representation: KSs may operate only on a few classes of blackboard objects
- Fully General Representation: All aspects of blackboard are understood by KSs

Proper balance between the two representations is necessary

Blackboard Systems versus Rule-based System

- The KSs are substantially larger and more complex than each isomorphic rule in an ES
 - Expert Systems work by firing a rule in response to a query, while a blackboard system works by executing an entire KS in response to an event
 - Each KS can be arbitrarily complex and internally different from one another. A single KS may be implemented as a complete rule-based system
- 

Truth Maintenance Systems (TMS)

- Truth Maintenance System is a structure which helps in revising sets of beliefs and maintaining the truth every time a new information contradicts information already present in the system
- TMS maintains the beliefs for general problem solving systems
- All TMS manipulate proposition symbols and the relationships between different proposition symbols
- Two types of TMS are
 - Monotonic TMS - It manipulates proposition symbols and Boolean constraints
 - Non-monotonic TMS – It allows for heuristic or non-monotonic relationships between proposition symbols such as ‘whenever P is true Q is likely’ or ‘if P is true then unless there is evidence to the contrary Q is assumed to be true’
- The inference engine explores alternatives, makes choices, and examines the consequences of the choices
- If a contradiction is noticed during this process, the TMS eliminates it by revising the knowledge base
- Both TMS and inference engine can together solve problems within large search spaces and in which algorithmic solutions do not exist
- TMS can be implemented explicitly in search problem-solving tools or implicitly within applications that solve particular search problems

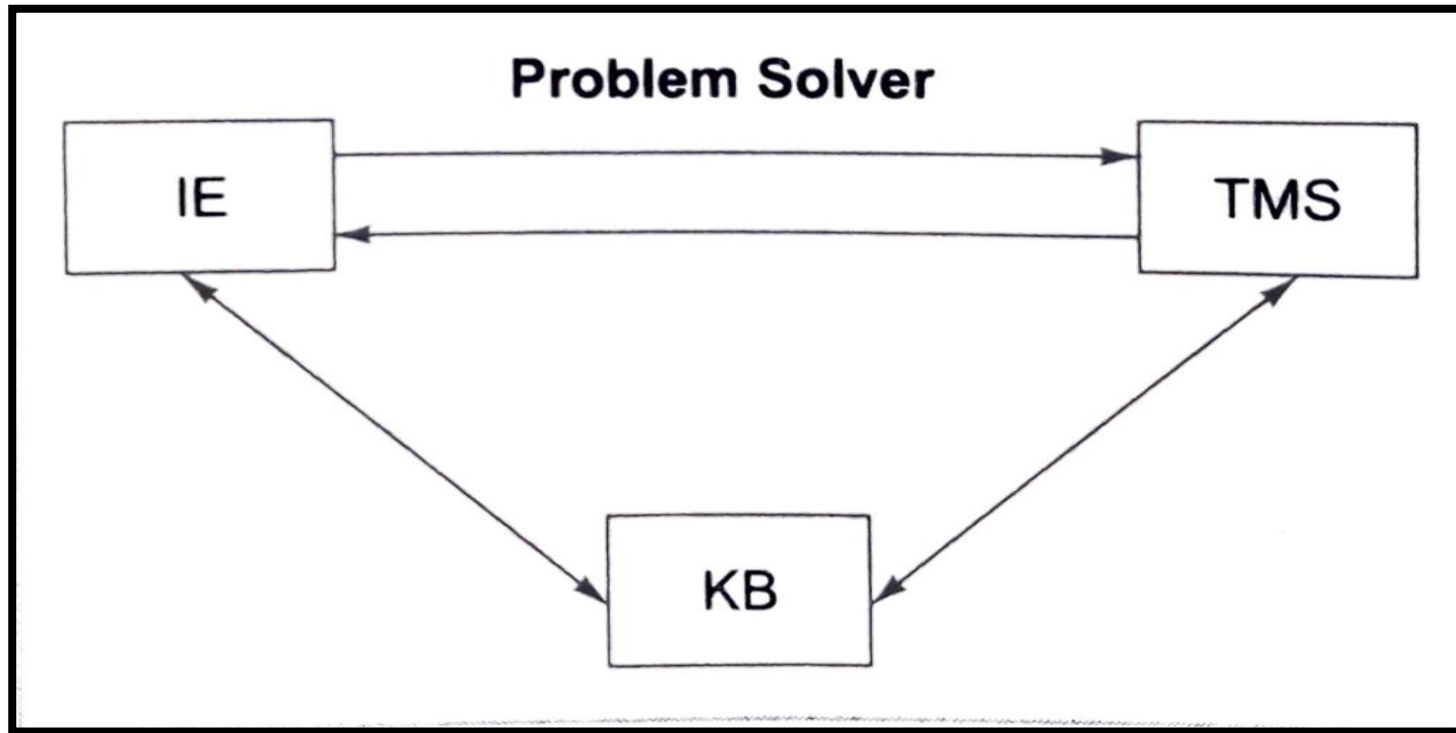
Monotonic System and Logic

- In monotonic systems, once a fact or piece of knowledge stored in the knowledge base is identified, it cannot be changed during the process of reasoning i.e. axioms are not allowed to change as once a fact is confirmed to be true, it must always remain true and can never be modified
- All the axioms used are either basic (or atomic) or can be derived from other facts that are also known to be true
- Making assumptions (such as probably, likely, etc.) are not allowed
- **Formal Definition: If a formula is a theorem for a particular formal theory, then that formula remains a theorem for any augmented theory obtained by adding axioms to the theory**
- For instance, if a property P is a theorem of T and if T is augmented to T_1 by additional axioms, then P remains a theorem of T_1
- Further, if an axiom A is added to theory T to build a theory T_1 , then all of the theorems of T are also theorems of T_1
- Similarly, if a proposition x can be derived from a set of propositions X and X is a subset of X_1 , then x can also be derived from X_1
- In monotonic reasoning, the world of axioms continually increases in size and keeps on expanding
- An example of monotonic form of reasoning is Predicate Logic. It represents a deductive reasoning system where new facts are derived from known facts

Non-monotonic System and Logic

- In non-monotonic systems, truths that are present in the system can be retracted whenever contradictions arise. Hence, number of axioms can increase as well as decrease
- The system is continually updated depending upon the changes in knowledge base
- In non-monotonic systems, the addition of an assertion of a belief to a theory can violate conclusions which have already been made
- In non-monotonic logic, if a formula is a theorem for a formal theory, then it need not be theorem for an augmented theory
- Common-sense reasoning is an example of non-monotonic reasoning. Non-monotonic reasoning is based on inferences made by applying non-monotonic logic

TMS



Interaction between different components in a Problem Solver

IE: Inference Engine
KB: Knowledge Base
TMS: Truth Maintenance System

Functionality of a Monotonic TMS

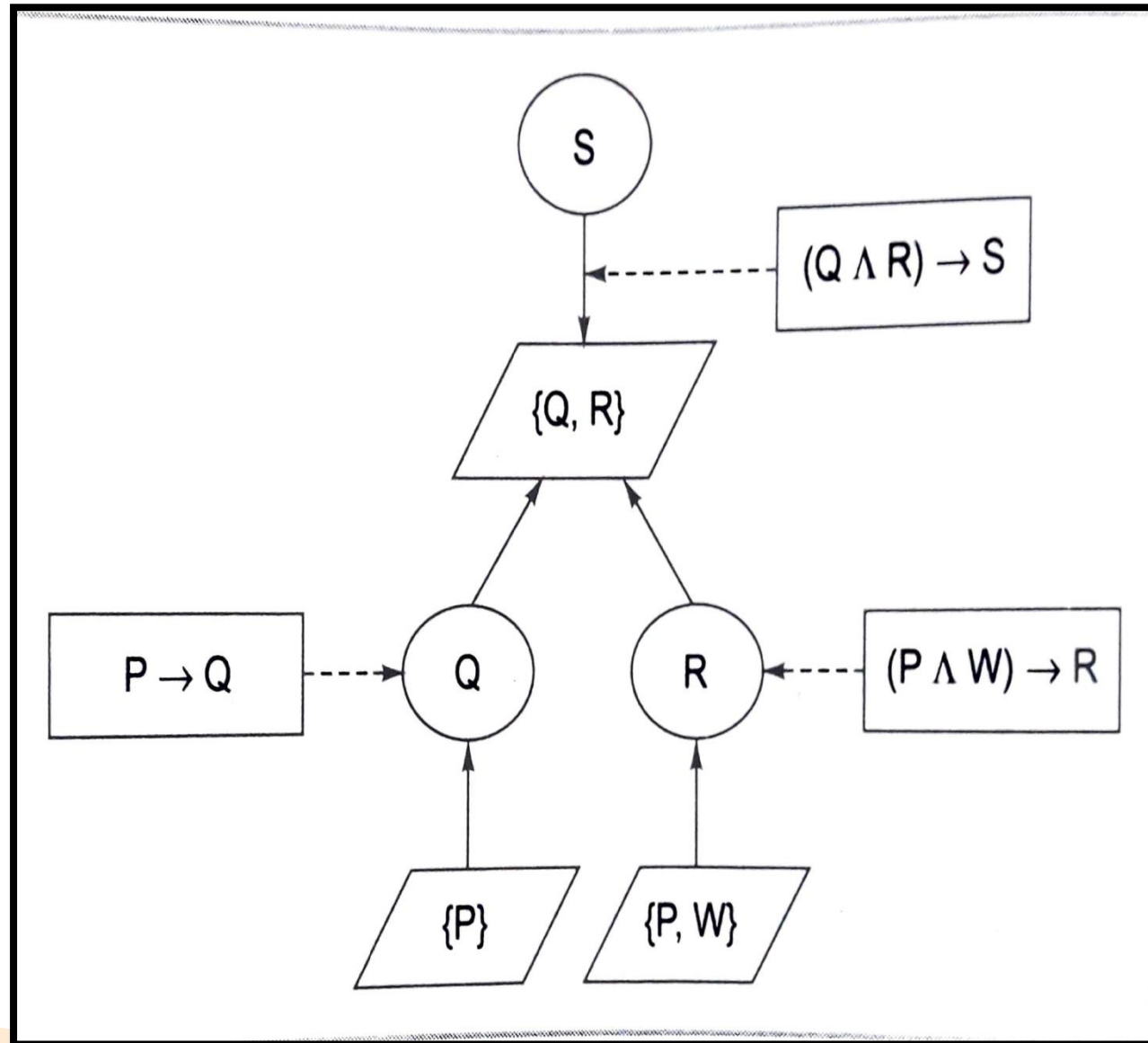
- The most practical applications of monotonic systems using TMS are qualitative simulation, fault diagnosis and search applications
- They provide a solid foundation upon which other kinds of systems could be built
- Algorithms for monotonic systems can usually be used in non-monotonic systems as well but the converse might not hold true
- A monotonic TMS is a general facility for manipulating Boolean constraints on proposition symbols
- A TMS can be used to ask questions about the consequences of the observations
- It consists of following generic interface functions:

- Add_constraint
- Follow_from
- Interface Functions

Justifying Literals	Derived Literals	Justifying constraints
{P, W}	R	$(P \wedge W) \rightarrow R$
{P}	Q	$P \rightarrow Q$
{Q,R}	S	$(Q \wedge R) \rightarrow S$

Justification Functions

Functionality of a Non-monotonic TMS



Justification Tree

Contradiction Handling

- The TMS interface function enables contradictory information to be given to the TMS.
- Most TMSs possess a technique of informing the user that a given premise set is inconsistent with the internal constraints
- This can be done by adding a special proposition symbol called Contradiction
- If the TMS is able to determine that a given premise set Σ contradicts the internal constraints then the function `follows_from(contradiction, Σ)` returns **yes**. In this case, `justifying_literals(contradiction, Σ)` and `justifying_constraints(contradiction, Σ)` return a set of literals and a set of constraints respectively, that underlie the contradiction

Non-monotonic TMS

- The basic operation of a TMS is to attach a justification to a fact
- A fact can be linked with any component of program knowledge which is to be connected with other components of program information
- A fact might be connected with each assertion and rule in a database, or might be attached with different meanings to various subsystem structures
- On the basis of justifications attached to the facts, TMS decides which beliefs in the truth of facts are supported by the recorded justifications

SL: Support List

SL(IN-node, OUT-node)


IN means that the belief is true

OUT-nodes do not support the considered node as true

Node number	Facts/Assertions	Justification(justified belief)
1	It is sunny	SL(3)(2,4)
2	It rains	SL()()
3	It is warm	SL(1)(2)
4	It is night time	SL()(1)

Justification of facts

Applications of TMS to search

- TMS is also useful in controlling searches, specifically those searches needed in Constraint Satisfaction problems
 - TMS organizes data within a data abstraction called a context, which corresponds to a single problem state and contains currently believed data
 - TMS provides believed data retrieval, belief revision, contradiction handling, and non-monotonicity handling, these features help a problem solver to examine state spaces
- 

Application of Expert Systems

The major purpose of building an ES for an organization is to preserve the know-how, experience, and expertise of the experts, which is a valuable asset to the organization. The purpose of ES is to provide this knowledge to other members of the organization for problem-solving purposes. Expert Systems have been widely developed and used to solve problems in different types of domains. Few of them are:

Diagnosis

The ES belonging to this class perform the task of inferring malfunctioning of system from observations. Such expert systems use situation descriptions, behavior characteristics, or knowledge about component design to determine the probable cause of system malfunction. Diagnosis can refer to inferring a possible disease from a given set of symptoms in the field of medicine.

Planning and Scheduling

The expert systems of this class help in designing actions and plans before actually solving a given problem. They analyze a set of one or more potentially complex and interacting goals in order to determine a set of actions that are needed to achieve these goals. Examples of this class are

- Airline scheduling of flights
- Manufacturing job-shop scheduling
- Creating plan for applying series of chemical reactions
- Manufacturing process planning
- Creating air-strike plan projected over several days, etc.


Design and manufacturing

This is one of the most important areas of ES applications. Here, a solution to a problem is configured by a given set of objects under a set of constraints. Configuration applications were initiated by computer companies to facilitate the manufacturing of semi-custom minicomputers. Examples of this are:

- Gene cloning
- Integrated circuits layouts designing
- Creation of complex organic molecules
- Modular home building
- Manufacturing, etc.

Prediction

The ES of this class perform the task of inferring the likely consequences of a given situation. For example:

- Weather prediction for rains, storms and tornado
 - Prediction of crops
 - Share market, etc.
- 

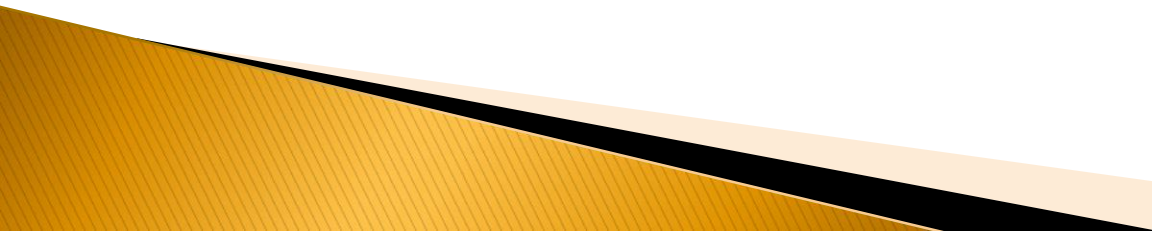
Interpretation

The expert systems of this class perform the task of interpreting and inferring situation description data of any domain such as geological data, census data, medical data , etc. For example

- Interpreting data from ICU test equipment for further investigation or monitoring
- Interpreting intelligent sensors reports to perform situation assessment
- Interpreting radar images to classify ships, etc.

Financial decision making

The financial services industry has been a prominent user of ES techniques. Such systems assist insurance companies to assess the risk presented by the customer and to determine a price for the insurance. Bankers use expert systems for assistance in determining whether to grant loans to business and individuals. A typical applications are:

- Foreign exchange trading
 - Formulating financial strategies
 - Giving advices, etc.
- 

Process monitoring and control

ES belonging to this class analyze real-time data from physical devices with the goal by comparing observations to expected outcomes, predicting trends, and controlling for both optimality and failure correction. Examples are:


- Steel-making and oil-refining industries
- Nuclear reactor(to detect accident conditions)
- Patient monitoring system in hospitals.

Instruction

An ES can offer tutorials and instructions to students by incorporating a student's behavior and learning capability models and can also evaluate student's acquired skills

Debugging

The systems of this class prescribe remedies for malfunctioning devices. They are

- Suggest the type of maintenance needed to correct faulty equipment.
 - Help in debugging large computer programs to improve the performance, etc.
- 

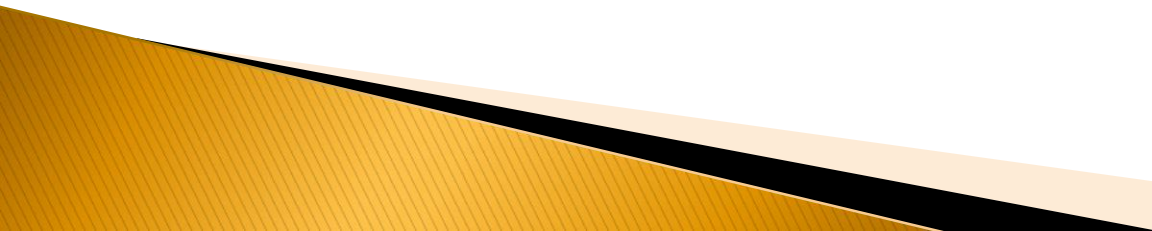
Knowledge publishing

This is relatively new but a potentially important area now-a-days. The primary function of ES in this field is to deliver knowledge to user that is in context of the user's problem. For example:

- Advisors, which counsel a user on appropriate grammatical usage in a text
- A tax advisor that accompanies a tax preparation program, which advises the user regarding tax strategy, tactics, and individual tax policy

Other Miscellaneous Applications

There exists a number of other applications where expert systems can be built. For eg.

- Fraud detection, object identification, information retrieval system
 - Handling certain tedious and dangerous situations such as coal mining
 - Judicial systems to solve new cases intelligently using past case history, advising on legal issues, etc.
 - In defense , performing situation assessment from intelligence reports, analyzing radar signals and images, predicting when and where armed conflict will occur next, simulation of war, etc.
 - Household activities such as giving advice on cooking, shopping, and so on are also potential applications
- 

Textbook:

1. Artificial Intelligence, Saroj Kaushik, 1st Edition, Cengage Learning

Reference Books:

1. Artificial Intelligence, Elaine Rich, Kevin Knight, Shivashankar B Nair, 3rd Edition, Tata McGraw Hill Education Private Limited., 2009
2. Artificial Intelligence- A modern Approach, 3rd Edition, Stuart Russel, Peter Norvig, Pearson Education