CS 332 Artificial Intelligence

SCHOOL OF COMPUTER ENGINEERING AND TECHNOLOGY

CS 332 Artificial Intelligence

Teaching Scheme **Credits: 2 + 1 = 3**

Theory: 4 Hrs / Week **Practical:** 2 Hrs / Week

Course Objectives:

1) To understand the concept of Artificial Intelligence (AI)
2) To learn various peculiar search strategies for AI
3) To develop a mind to solve real world problems unconventionally with optimality

Course Outcomes:

1) Identify and apply suitable Intelligent agents for various AI applications
2) Design smart system using different informed search / uninformed search or heuristic approaches.
3) Identify knowledge associated and represent it by ontological engineering to plan a strategy to solve given problem.

Syllabus

Knowledge Inference and Expert System

Basics of Probability, Markov Model, Statistical reasoning, Bayes' Theorem and its use, Bayesian learning and network

Expert systems: Architecture of Expert system, Role of Expert system, Inference engine, Knowledge acquisition, Typical Expert systems- MYCIN, Expert systems shells, Applications of Expert systems.

Knowledge Inference and Expert Systems

Contents

- Basics of Probability,
- Markov Model,
- Statistical reasoning,
- Bayes' Theorem and its use, Bayesian learning and network
- Architecture of Expert system, Role of Expert system,
- Inference engine, Knowledge acquisition,
- Typical Expert systems- MYCIN, Expert systems shells,
- Applications of Expert systems.

Acting under uncertainty

A logical agent uses propositions that are true, false or unknown

When the logical agent knows enough facts about it environment it derives plans that are guaranteed to work.

Unfortunately, agents almost never have access to the whole truth about their environment and therefore **agents must act under uncertainty**

Uncertainty

Let action A_t = leave for airport t minutes before flight . Will A_t get me there on time?

Problems:

1. partial observability (road state, other drivers' plans, noisy sensors)
2. uncertainty in action outcomes (flat tire, etc.)
3. immense complexity of modeling and predicting traffic

Hence a purely logical approach either

1. risks falsehood: “A_{25} will get me there on time”, or
2. leads to conclusions that are too weak for decision making:

“A_{25} will get me there on time if there's no accident on the bridge and it doesn't rain and my tires remain intact etc etc.” (A_{1440} might reasonably be said to get me there on time but I'd have to stay overnight in the airport …)

Probability to the Rescue

Probability

- Model agent's degree of belief, given the available evidence.

- A_{25} will get me there on time with probability 0.04

Probability in AI models our ignorance, not the true state of the world.

The statement "With probability 0.7 I have a cavity" means:
I either have a cavity or not, but I don't have all the necessary information to know this for sure.

Probability

1. All probability statements must indicate the evidence with respect to which the probability is being assessed
2. As agent receives new percepts, the probability assessments are updated to reflect new evidence
3. Before the evidence is obtained , we have **prior** or **unconditional probability**
4. After the evidence is obtained, we have **posterior or conditional probability**
5. Generally, the **agent** will have some evidence from its percepts and will be interested in computing the posterior probabilities of the outcomes it cares about

Probability

Subjective probability:

Probabilities relate propositions to agent's own state of knowledge

e.g., $P(A_{25} \mid \text{no reported accidents at 3 a.m.}) = 0.06$

Probabilities of propositions change with new evidence:

e.g., $P(A_{25} \mid \text{no reported accidents at 5 a.m.}) = 0.15$

Making decisions under uncertainty

Suppose I believe the following:

$P(A_{25}$ gets me there on time | …) = 0.04

$P(A_{90}$ gets me there on time | …) = 0.70

$P(A_{120}$ gets me there on time | …) = 0.95

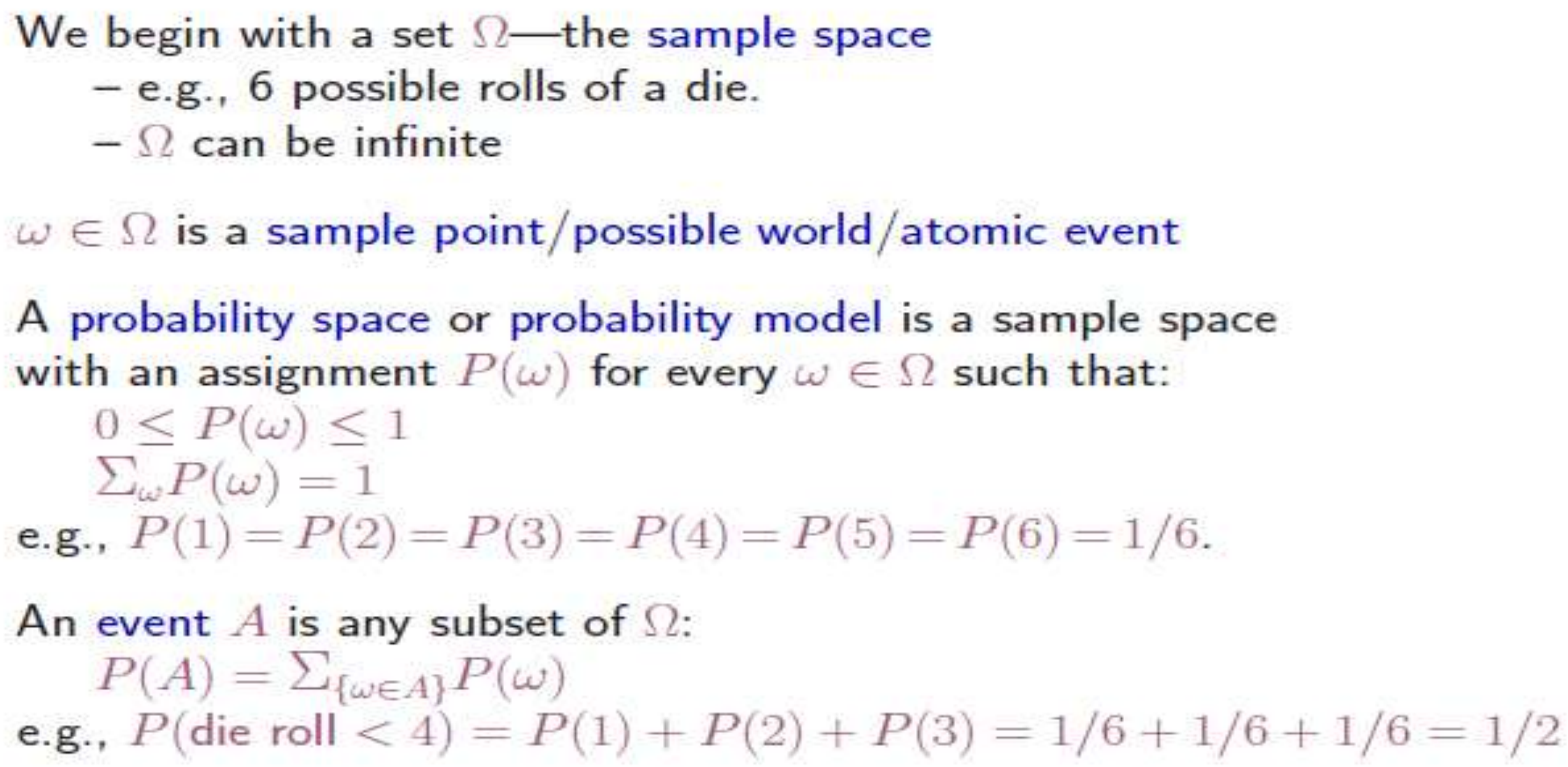
$P(A_{1440}$ gets me there on time | …) = 0.9999

Which action to choose?

Depends on my preferences for missing flight vs. time spent waiting, etc.

- Utility theory is used to represent and infer preferences
- Decision theory = probability theory + utility theory

Probability Basics



We begin with a set Ω—the sample space

– e.g., 6 possible rolls of a die.

– Ω can be infinite

$\omega \in \Omega$ is a sample point/possible world/atomic event

A probability space or probability model is a sample space with an assignment $P(\omega)$ for every $\omega \in \Omega$ such that:

$0 \leq P(\omega) \leq 1$

$\Sigma_\omega P(\omega) = 1$

e.g., $P(1) = P(2) = P(3) = P(4) = P(5) = P(6) = 1/6.$

An event A is any subset of Ω:

$P(A) = \Sigma_{\{\omega \in A\}} P(\omega)$

e.g., $P(\text{die roll} < 4) = P(1) + P(2) + P(3) = 1/6 + 1/6 + 1/6 = 1/2$

Probability Basics

- A random variable is a function from sample points to some range
  - e.g., $Odd(1) = true$, has a boolean-valued range.

P induces a probability distribution for any r.v. X:

$$P(X = x_i) = \Sigma_{\{\omega : X(\omega) = x_i\}} P(\omega)$$

e.g., $P(Odd = true) = P(1) + P(3) + P(5) = 1/6 + 1/6 + 1/6 = 1/2$

Capital letter: random variable
lower case: single value

Probability Basics

Basic element: random variable

Similar to propositional logic: possible worlds defined by assignment of values to random variables.

Boolean random variables
 e.g., *Cavity* (do I have a cavity?)

Discrete random variables
 e.g., *Weather* is one of <*sunny*,*rainy*,*cloudy*,*snow*>

Elementary proposition constructed by assignment of a value to a random variable: e.g., *Weather* = *sunny*, *Cavity* = *false* (abbreviated as $\neg$*cavity*)

Complex propositions formed from elementary propositions and standard logical connectives e.g., *Weather* = *sunny* $\vee$ *Cavity* = *false*

Syntax

Atomic event: A complete specification of the state of the world about which the agent is uncertain (i.e. a full assignment of values to all variables in the universe, a unique single world).

E.g., if the world consists of only two Boolean variables *Cavity* and *Toothache*, then there are 4 distinct atomic events:

$Cavity = false \wedge Toothache = false$

$Cavity = false \wedge Toothache = true$

$Cavity = true \wedge Toothache = false$

$Cavity = true \wedge Toothache = true$

$\downarrow$ $\downarrow$

if some atomic event is true, then all other other atomic events are false.

There is always some atomic event true.

Hence, there is exactly 1 atomic event true.

Atomic events are mutually exclusive and the set of all possible atomic events is exhaustive

Prior probability

Prior or unconditional probabilities of propositions
e.g., P(*Cavity* = true) = 0.1 and P(*Weather* = sunny) = 0.72 correspond to belief prior to arrival of any (new) evidence

Probability distribution gives values for all possible assignments:
P(*Weather*) = <0.72,0.1,0.08,0.1> (normalized, i.e., sums to 1)

Joint probability distribution for a set of random variables gives the probability of every atomic event of those random variables
P(*Weather*,*Cavity*) = a 4 × 2 matrix of values:

Weather =	sunny	rainy	cloudy	snow
Cavity = true	0.144	0.02	0.016	0.02
Cavity = false	0.576	0.08	0.064	0.08

Every question about a domain can be answered by the joint distribution

Conditional probability

Conditional or posterior probabilities
e.g., P(*cavity* | *toothache*) = 0.8 i.e., given that *Toothache*=*true* is all I know.

Note that **P**(Cavity|Toothache) is a 2x2 array, normalized over columns.

If we know more, e.g., cavity is also given, then we have
P(*cavity* | *toothache*,*cavity*) = 1

New evidence may be irrelevant, allowing simplification, e.g.,
P(*cavity* | *toothache*, *sunny*) = P(*cavity* | *toothache*) = 0.8

Conditional probability

Definition of conditional probability:

$P(a \mid b) = P(a \wedge b) / P(b)$ if $P(b) > 0$

Product rule gives an alternative formulation:

$P(a \wedge b) = P(a \mid b) P(b) = P(b \mid a) P(a)$

Bayes Rule: $P(a|b) = P(b|a) P(a) / P(b)$

A general version holds for whole distributions, e.g.,

$\mathbf{P}(Weather, Cavity) = \mathbf{P}(Weather \mid Cavity)\, \mathbf{P}(Cavity)$

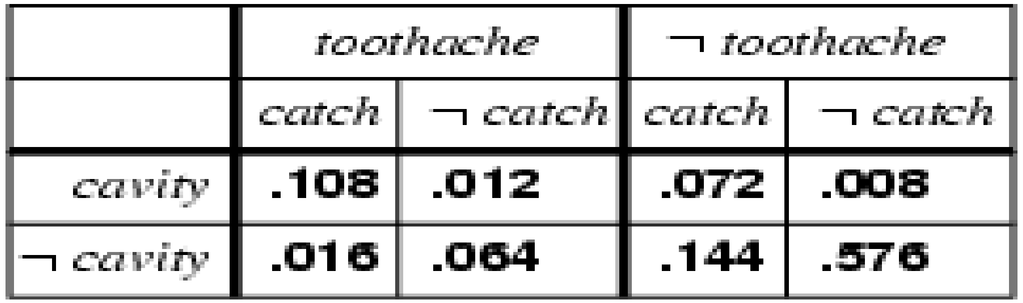
(View as a set of 4 × 2 equations, not matrix multiplication)

Chain rule is derived by successive application of product rule:

$$\begin{aligned}\mathbf{P}(X_1, \ldots, X_n) &= \mathbf{P}(X_1,\ldots,X_{n-1})\, \mathbf{P}(X_n \mid X_1,\ldots,X_{n-1}) \\ &= \mathbf{P}(X_1,\ldots,X_{n-2})\, \mathbf{P}(X_{n-1} \mid X_1,\ldots,X_{n-2})\, \mathbf{P}(X_n \mid X_1,\ldots,X_{n-1}) \\ &= \ldots \\ &= \pi_{i=1}\text{^}n\, \mathbf{P}(X_i \mid X_1, \ldots, X_{i-1})\end{aligned}$$

Inference by enumeration

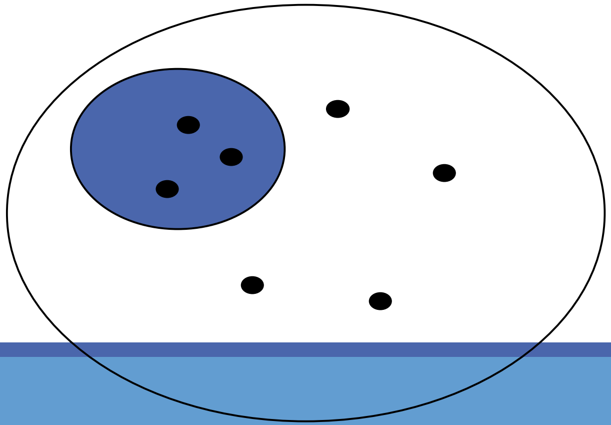
Start with the joint probability distribution:



	toothache		¬ *toothache*	
	catch	¬ *catch*	*catch*	¬ *catch*
cavity	.108	.012	.072	.008
¬ *cavity*	.016	.064	.144	.576

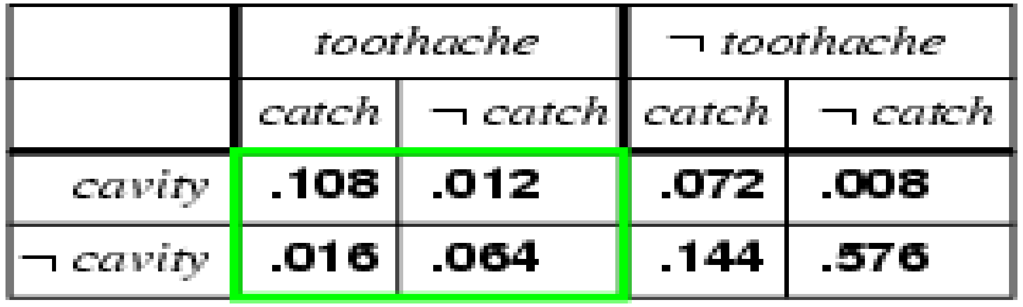
For any proposition a, sum the atomic events where it is true: $P(a) = \Sigma_{\omega \text{ s.t. } a=\text{true}} P(\omega)$

P(a)=1/7 + 1/7 + 1/7 = 3/7



Inference by enumeration

Start with the joint probability distribution:



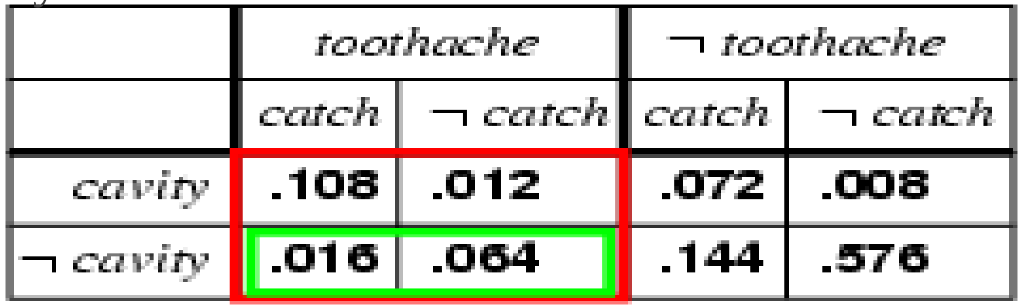
	toothache		*¬ toothache*	
	catch	*¬ catch*	*catch*	*¬ catch*
cavity	.108	.012	.072	.008
¬ cavity	.016	.064	.144	.576

For any proposition a, sum the atomic events where it is true: $P(a) = \Sigma_{\omega:\omega \text{ s.t. } a=\text{true}} \; P(\omega)$

P(*toothache*) = 0.108 + 0.012 + 0.016 + 0.064 = 0.2

Inference by enumeration

Start with the joint probability distribution:



	toothache		¬ *toothache*	
	catch	¬ *catch*	*catch*	¬ *catch*
cavity	.108	.012	.072	.008
¬ *cavity*	.016	.064	.144	.576

Can also compute conditional probabilities:

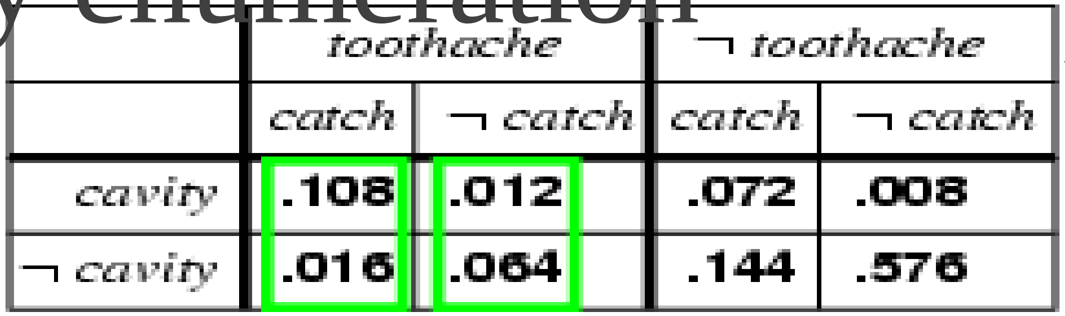
$P(\neg cavity \mid toothache) = P(\neg cavity \wedge toothache) / P(toothache)$

$= (0.016+0.064) / (0.108 + 0.012 + 0.016 + 0.064)$

$= 0.4$

Calculate (i) $P(cavity)$ (ii) $P(cavity \mid toothache)$

Inference by enumeration



	toothache		¬ toothache	
	catch	¬ catch	catch	¬ catch
cavity	.108	.012	.072	.008
¬ cavity	.016	.064	.144	.576

Denominator can be viewed as a normalization constant α

P(*Cavity* | *toothache*) = α x **P**(*Cavity*,*toothache*)
= α x [**P**(*Cavity*,*toothache*,*catch*) + **P**(*Cavity*,*toothache*,¬ *catch*)]
= α x [<0.108,0.016> + <0.012,0.064>]
= α x <0.12,0.08> = <0.6,0.4>

General idea: compute distribution on query variable by fixing evidence variables and summing over hidden variables

Inference by enumeration

Typically, we are interested in

the posterior joint distribution of the query variables **Y**

given specific values **e** for the evidence variables **E**

Let the hidden variables be $\mathbf{H} = \mathbf{X} - \mathbf{Y} - \mathbf{E}$

Then the required summation of joint entries is done by summing out the hidden variables:

$$\mathbf{P}(\mathbf{Y} \mid \mathbf{E} = \mathbf{e}) = \alpha \mathbf{P}(\mathbf{Y}, \mathbf{E} = \mathbf{e}) = \alpha \Sigma_{\mathrm{h}} \mathbf{P}(\mathbf{Y}, \mathbf{E} = \mathbf{e}, \mathbf{H} = \mathbf{h})$$

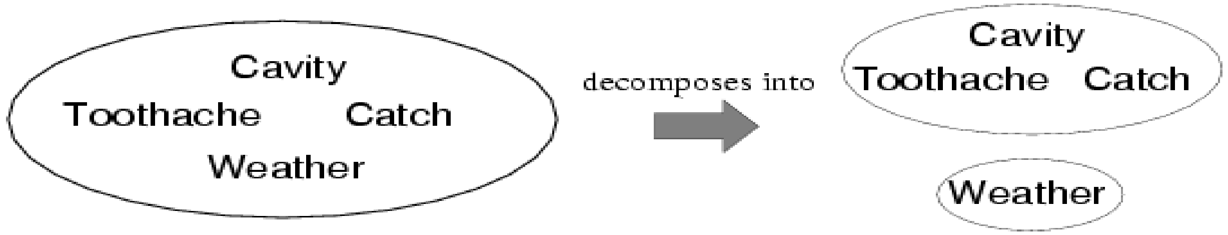
The terms in the summation are joint entries because **Y**, **E** and **H** together exhaust the set of random variables

Obvious problems:

1. Worst-case time complexity $O(d^n)$ where d is the largest arity
2. Space complexity $O(d^n)$ to store the joint distribution
3. How to find the numbers for $O(d^n)$ entries

Independence

A and *B* are independent iff $\mathbf{P}(A|B) = \mathbf{P}(A)$ or $\mathbf{P}(B|A) = \mathbf{P}(B)$ or $\mathbf{P}(A, B) = \mathbf{P}(A)\,\mathbf{P}(B)$



$\mathbf{P}(Toothache, Catch, Cavity, Weather)$
$= \mathbf{P}(Toothache, Catch, Cavity)\,\mathbf{P}(Weather)$

32 entries reduced to 12;

for *n* independent biased coins, $O(2^n) \rightarrow O(n)$

Absolute independence powerful but rare

Dentistry is a large field with hundreds of variables, none of which are independent. What to do?

Bayes' Rule

Product rule $P(a \wedge b) = P(a \mid b) P(b) = P(b \mid a) P(a)$

$\Rightarrow$ Bayes' rule: $P(a \mid b) = P(b \mid a) P(a) / P(b)$

or in distribution form

$$\mathbf{P}(Y|X) = \mathbf{P}(X|Y)\, \mathbf{P}(Y) / \mathbf{P}(X) = \alpha \mathbf{P}(X|Y)\, \mathbf{P}(Y)$$

- E.g., let m be meningitis, s be stiff neck: Let $P(s \mid m)=0.8$, $P(m)=0.0001$, $P(s)=0.1$

  $P(m|s) = P(s|m)\, P(m) / P(s) = 0.8 \times 0.0001 / 0.1 = 0.0008$

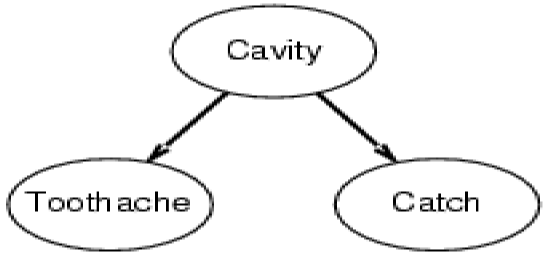
- Note: even though the probability of having a stiff neck given meningitis is very large (0.8), the posterior probability of meningitis given a stiff neck is still very small

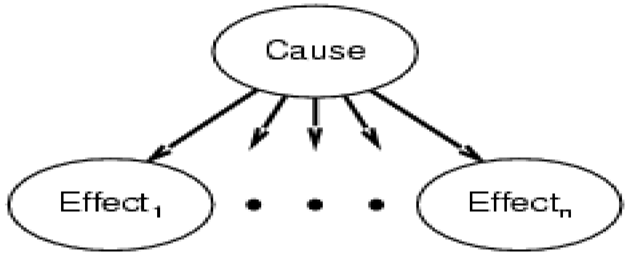
- $P(s|m)$ is more ‘robust’ than $P(m|s)$. Imagine a new disease appeared
  which would also cause a stiff neck, then $P(m|s)$ changes but $P(s|m)$ not.

Bayes' Rule and conditional independence

This is an example of a naïve Bayes model:

$\mathbf{P}(\text{Cause},\text{Effect}_1, \ldots ,\text{Effect}_n) = \mathbf{P}(\text{Cause})\ \pi_i \mathbf{P}(\text{Effect}_i|\text{Cause})$





Total number of parameters is linear in *n*

A naive Bayes classifier computes: P(cause|effect1, effect2...)

Graphical Models

If no assumption of independence is made, then an exponential number of parameters must be estimated for sound probabilistic inference.

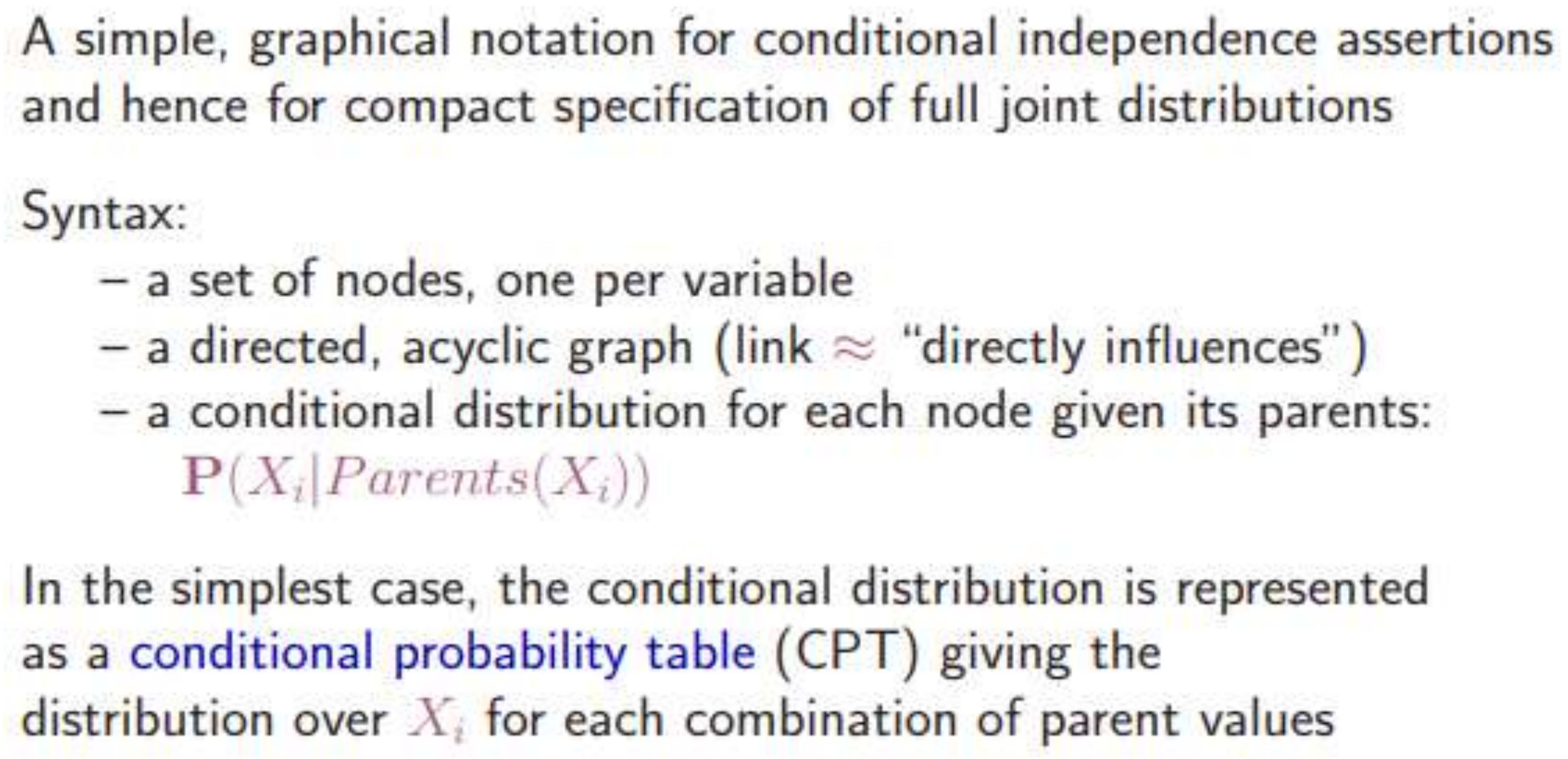
No realistic amount of training data is sufficient to estimate so many parameters.

If a blanket assumption of conditional independence is made, efficient training and inference is possible, but such a strong assumption is rarely warranted.

Graphical models use directed or undirected graphs over a set of random variables to explicitly specify variable dependencies and allow for less restrictive independence assumptions while limiting the number of parameters that must be estimated.

- **Bayesian Networks**: Directed acyclic graphs that indicate causal structure.
- **Markov Networks**: Undirected graphs that capture general dependencies.

Bayesian network



A simple, graphical notation for conditional independence assertions and hence for compact specification of full joint distributions

Syntax:

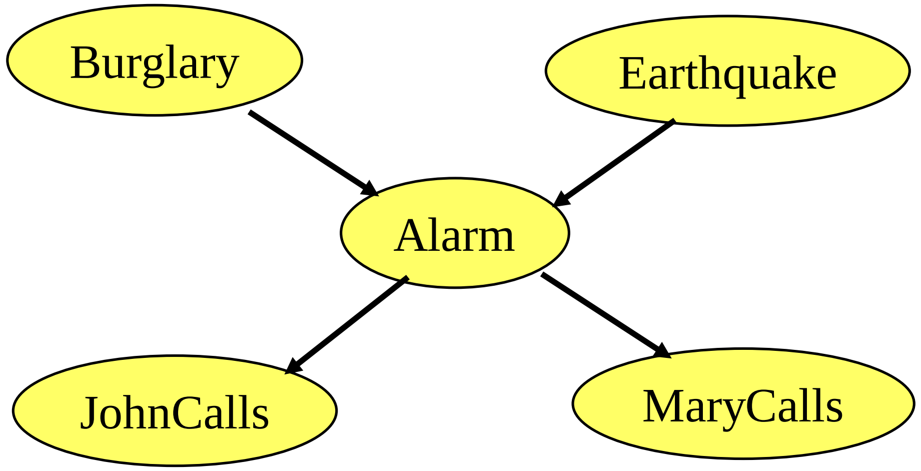
- a set of nodes, one per variable
- a directed, acyclic graph (link $\approx$ "directly influences")
- a conditional distribution for each node given its parents:
  $\mathbf{P}(X_i|Parents(X_i))$

In the simplest case, the conditional distribution is represented as a conditional probability table (CPT) giving the distribution over X_i for each combination of parent values

Bayesian Networks

Directed Acyclic Graph (DAG)

- Nodes are random variables
- Edges indicate causal influences



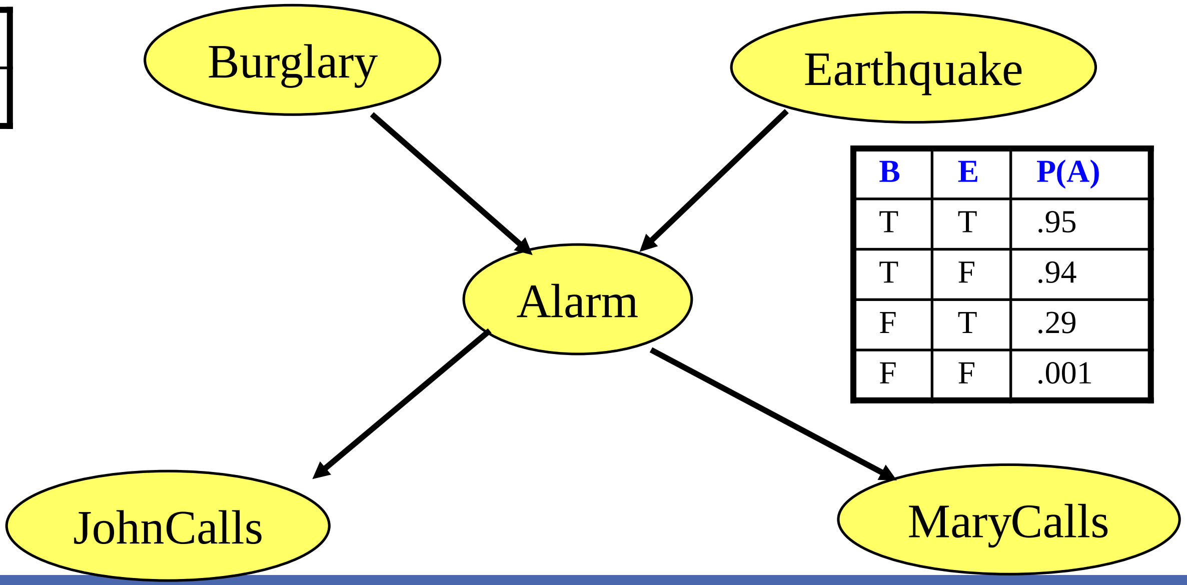
Conditional Probability Tables

Each node has a **conditional probability table** (**CPT**) that gives the probability of each of its values given every possible combination of values for its parents (conditioning case).

- Roots (sources) of the DAG that have no parents are given prior probabilities.

P(B)
.001

P(E)
.002



B	E	P(A)
T	T	.95
T	F	.94
F	T	.29
F	F	.001

A	P(J)
T	.90
F	.05

A	P(M)
T	.70
F	.01

Joint Distributions for Bayes Nets

A Bayesian Network implicitly defines a joint distribution.

$$P(x_1, x_2, \ldots x_n) = \prod_{i=1}^{n} P(x_i \mid \text{Parents}(X_i))$$

- Example

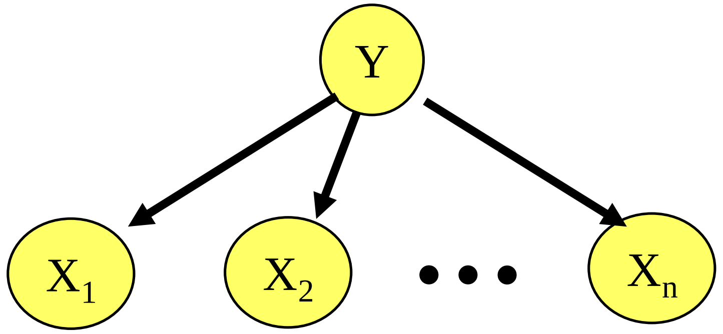
$$P(J \wedge M \wedge A \wedge \neg B \wedge \neg E)$$

$$= P(J \mid A) P(M \mid A) P(A \mid \neg B \wedge \neg E) P(\neg B) P(\neg E)$$

$$= 0.9 \times 0.7 \times 0.001 \times 0.999 \times 0.998 = 0.00062$$

Naïve Bayes as a Bayes Net

Naïve Bayes is a simple Bayes Net



- Priors P(*Y*) and conditionals P(X_i|*Y*) for Naïve Bayes provide CPTs for the network.

WHAT IS A MARKOV MODEL?

•A Markov Model is a stochastic model which models temporal or sequential data, i.e., data that are ordered.

•It provides a way to model the dependencies of current information (e.g. weather) with previous information.

•It is composed of states, transition scheme between states, and emission of outputs (discrete or continuous).

•Several goals can be accomplished by using Markov models:

Learn statistics of sequential data.

Do prediction or estimation.

Recognize patterns.

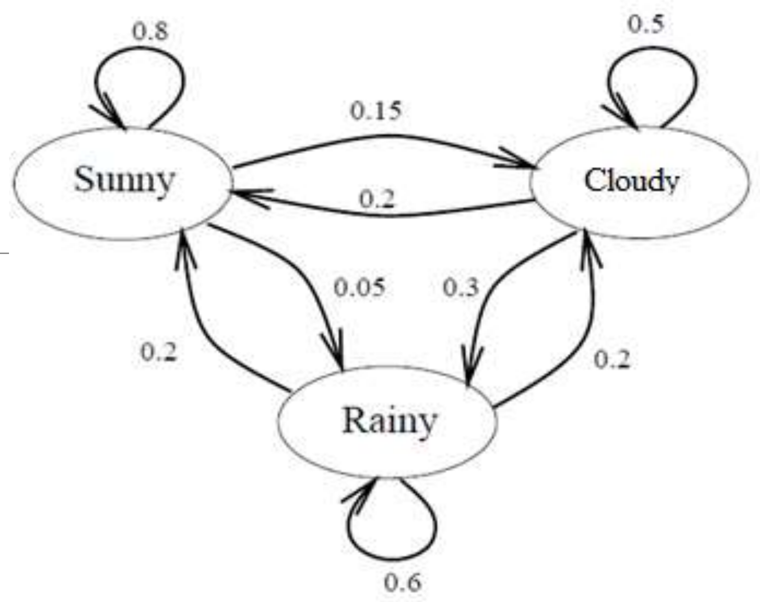
Markov Model Example

Design a Markov Model to predict the weather of tomorrow using previous information of the past days.

•Our model has only 3 states: $S = S1, S2, S3$, and the name of each state is $S1 = Sunny$, $S2 = Rainy$, $S3 = Cloudy$.

•To establish the transition probabilities relationship between states we will need to collect data.

Assume the data produces the following transition probabilities:



Markov model is represented by a graph with set of vertices corresponding to the set of states Q and probability of going from state i to state j is described by matrix

A which is n x n transition probability matrix
A(i,j)= P[q $_{t+1}$=j|q $_{t}$ =i] where q t denotes state at time t

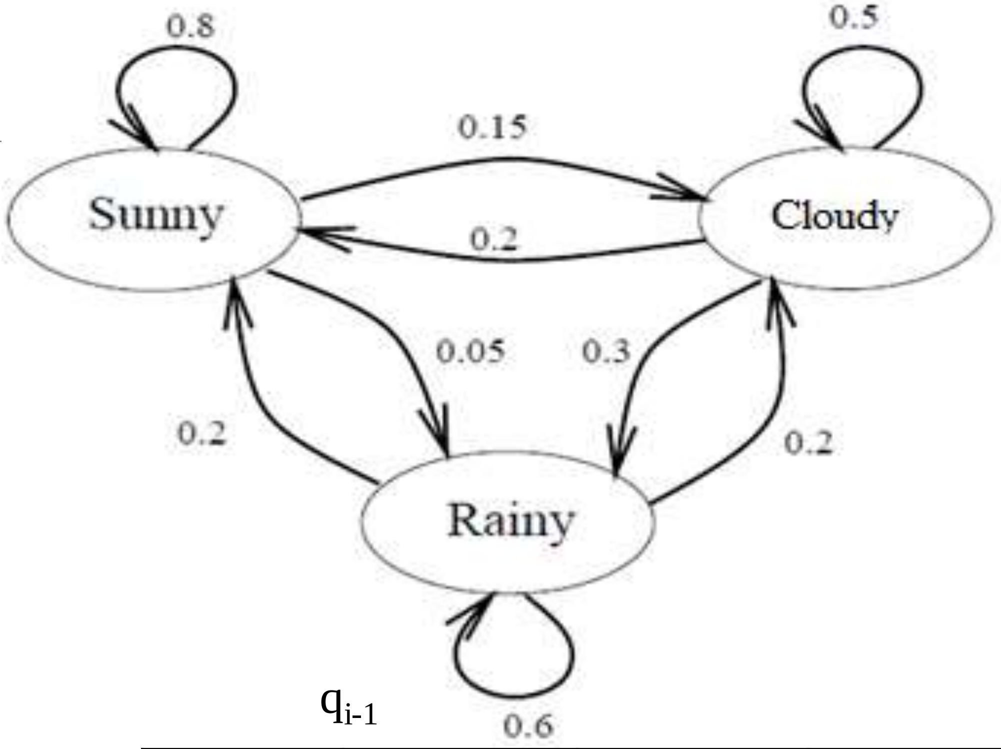
Thus Markov model M is described by set of states Q and a transition probability matrix A such that M = (Q, A)

Markov Models

$P(Sunny|Sunny)$=0.8
$P(Rainy|Sunny)$=0.05
$P(Cloudy|Sunny)$=0.15

$P(Sunny|Rainy)$=0.2
$P(Rainy|Rainy)$=0.6
$P(Cloudy|Rainy)$=0.2

$P(Sunny|Cloudy)$=0.2
$P(Rainy|Cloudy)$=0.3
$P(Cloudy|Cloudy)$=0.5



q_{i-1}

q_i

	Sunny	Rainy	Cloudy
Sunny	0.8	0.2	0.2
Rainy	0.05	.6	.3
Cloudy	0.15	0.2	0.5

Markov Models

Let’s say we have a sequence: Sunny, Rainy, Cloudy, Cloudy, Sunny, Sunny, Sunny, Rainy, ….; so, in a day we can be in any of the three states.

•We can use the following state sequence notation:

$q1$, $q2$, $q3$, $q4$, $q5$,.., where $qi \epsilon \{Sunny, Rainy, Cloudy\}$.

•In order to compute the probability of tomorrow’s weather we can use the Markov property:

$$P(q_1, \ldots, q_n) = \prod_{i=1}^{n} P(q_i | q_{i-1})$$

Markov Property

$$P(q_1, \dots, q_n) = \prod_{i=1}^{n} P(q_i | q_{i-1})$$

So, what is Markov property

• The probability of being in a specific state at a future time t only depends ONLY on the state of the system right now (s) state, and NOT at all about the states the system has had before (before s).

• A "memorylessness" process, where the next state of the process depends only on the previous state and not the sequence of states.

Markov Models

Exercise 1: Given that today is Sunny, what's the probability that tomorrow is Sunny and the next day Rainy?

$P(q_2,q_3|q_1)=P(q_2\,|\,q_1)\,P(q_3\,|\,q_1,q_2)$
$= P(q_2|q_1)\,P(q_3\,|\,q_2\,)$
$= P(Sunny\,|\,Sunny)\,P(Rainy\,|Sunny\,)$
$=0.8(0.05)$
$=0.04$

Exercise 2: Assume that yesterday's weather was Rainy, and today is Cloudy, what is the probability that tomorrow will be Sunny?

Expert System

Attempt to model expert decision making in a limited domain

Examples: medical diagnosis, computer configuration, machine fault diagnosis

Requires a willing Expert

Requires knowledge representable as rules

- Doesn’t work for chess

Preponderance of evidence for decision, not proof. (Civil law suits)

Rule Based System

What is a Rule Based System? Rule based system or knowledge based systems are specialized software that encapsulate ‘Human Intelligence’ like knowledge there by make intelligent decisions quickly and in repeatable form

Rule-based systems (RBS) provide automatic problem solving tools for capturing the human expertise and decision making.

Some of the RBS application areas are:

Equipment maintenance - diagnosing faults and recommending repairs

Component selection - eliciting requirements from electronics catalogs

Computer operation - analyzing requirements, and operating software

Product configuration - identifying parts that satisfy constraints

Troubleshooting - suggesting treatments, and prescribing preventative measures

Process control - spotting problematic data and removing irregularities

Quality assurance - assessing tasks, proposing practices

medical diagnosis

RBS has two *distinguishing characteristics*:

The existing knowledge can be refined and new knowledge can be added for incremental increase of the system performance.

They explain the line of reasoning making their logic transparent.

Properties of rule-based systems

They incorporate practical human knowledge in if-then rules.

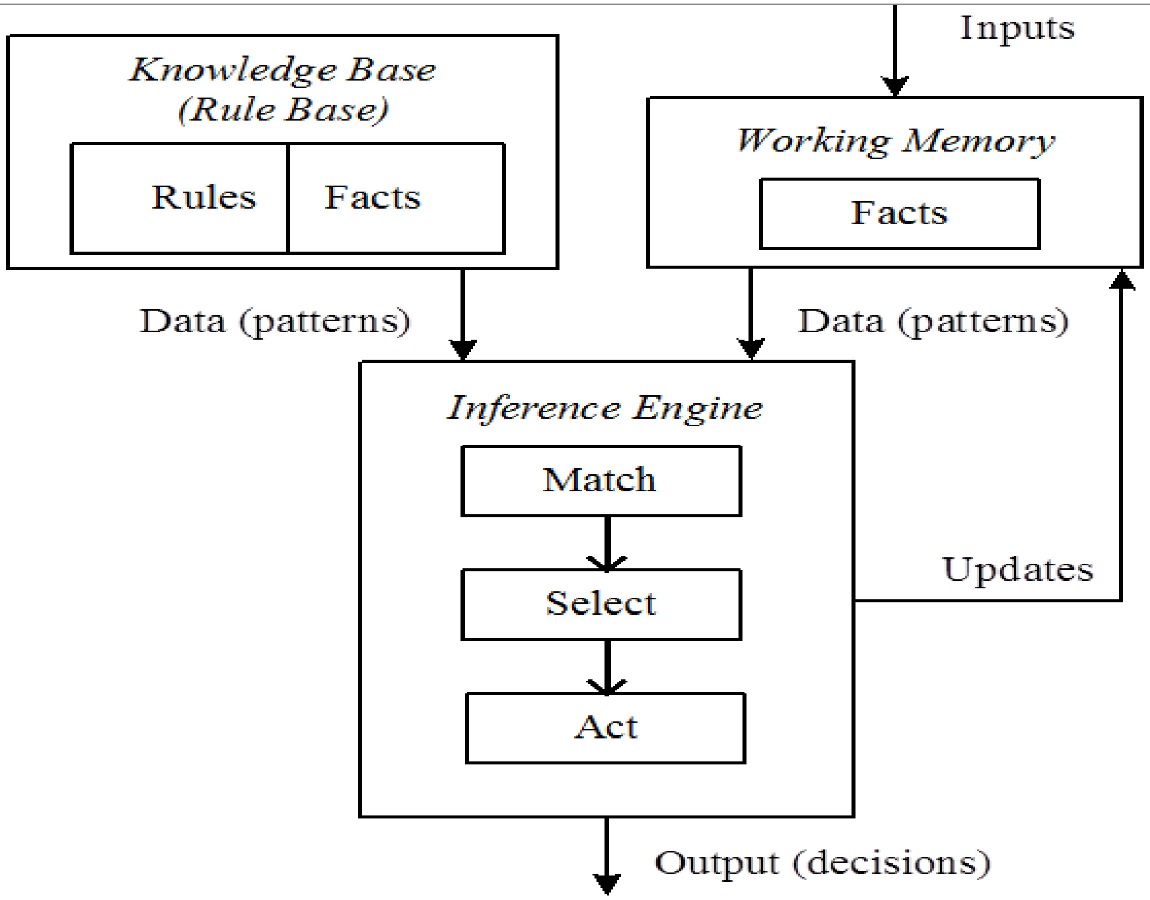
Their skill increases proportionally to the enlargement of the knowledge.

They can solve a wide range of potentially complex problems by selecting relevant rules and then combining the results.

They adaptively determine the best sequence of rules to examine.

They explain their conclusions by retracting their lines of reasoning.

Architecture of RBS



Continued…

An RBS consists of a ***knowledge base*** and an ***inference engine***.

The knowledge base contains *rules* and *facts*.

- **Rules**

The rules contain several if-patterns (antecedents) and one or more then-patterns (consequents).

The interpretation of a rule is that if the antecedent can be satisfied then the consequent is satisfied too.

When the consequent defines an action the effect of satisfying the antecedent is to schedule the action for execution. When the consequent defines a conclusion the effect is to infer the conclusion.

The rules specify chunks of analytic problem-solving knowledge. They use symbolic descriptions to characterize relevant situations and corresponding actions. The language used for these descriptions imposes a conceptual framework on the problem and its solutions.

- **Facts**

The *facts* express assertions about properties, relations, propositions, etc. In contrast to rules which the RBS interprets as imperatives, facts are usually static and inactive regarding the programmatic value of knowledge.

- **Working Memory**

- The *knowledge base* that contains the rules is long-term store.

In addition to this static memory, the RBS uses a *working memory*

(short-term store) to store temporary assertions.

These assertions record earlier rule-based inferences, thus the

contents of the working memory is problem-solving state information.

The working memory data adhere to the syntactic conventions for facts.

- **Inference Engine**

The basic function of the RBS is to produce outputs, which may be a problem solution, an answer to a question, or an analysis of some data.

The main operating cycle of RBS consists of three phases:

match, *select* and *act*.

When certain triggering (firing) conditions occur the actions specified by the rules are executed. The firing conditions are defined by the data patterns in the working memory.

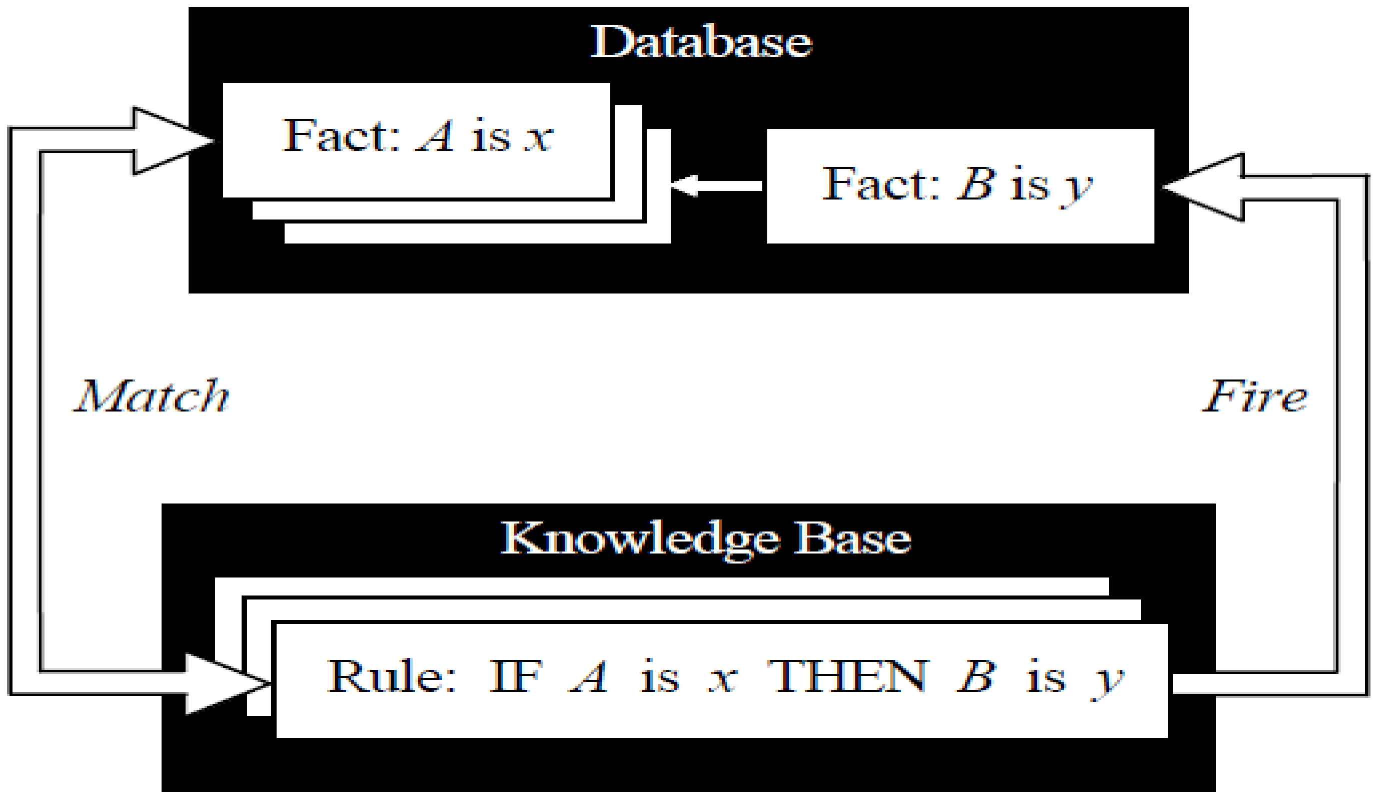
When the patterns in a rule are matched, the system selects this rule and interprets it so as to draw inferences that alter the working memory.

The most common RBS modes of operation are:

- forward chaining (stimulus driven)

- backward chaining (goal directed)

Inference engine cycles via a match-fire procedure



Inferencing

Given a set of rules like these, there are essentially two ways we can use them to generate new knowledge:

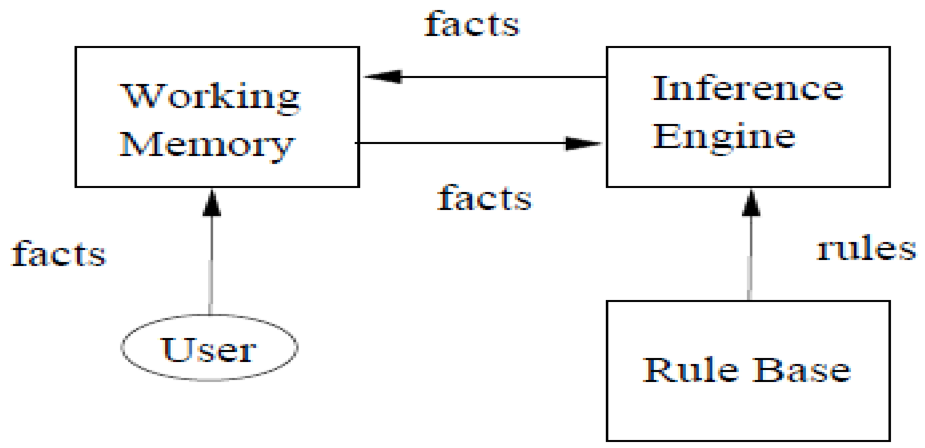
1. forward chaining: starts with the facts, and sees what rules apply (and hence what should be done) given the facts. data driven;

2. backward chaining : starts with something to find out, and

   looks for rules that will help in answering it goal driven.

In a forward chaining system: Facts are held in a working memory Condition-action rules represent actions to take when specified facts occur in working memory.

Typically the actions involve adding or deleting facts from working memory.


facts
Working Memory
Inference Engine
facts
facts
rules
User
Rule Base

***Forward Chaining* :** Forward chaining mode of operation means that a rule is triggered when changes in the working memory produce a situation that matches all of its antecedents.

Forward chaining is the process of inferring then-patterns from if-patterns, that is consequents from antecedents. When an antecedent matches an assertion the antecedent is satisfied.

When all antecedents of a rule are satisfied the rule is triggered. In deduction systems all triggered rules are allowed and may fire.

Forward Chaining Algorithm

1. *Repeat*
2. *For* each rule do
3. *Match* all its antecedents to the facts from the Working memory
4. - if all antecedents of a rule are matched,
5. *Execute* is consequents
6. *until* no rule produces a new assertion, or the goal is satisfied

Example: A deduction system for identification of animals:

```
(RULE 1  (IF   ( ?x has hair ))
         (THEN ( ?x is mammal )))
(RULE 2  (IF   ( ?x gives milk ))
         (THEN ( ?x is mammal )))
(RULE 3  (IF   ( ?x has feathers ))
         (THEN ( ?x is bird )))
(RULE 4  (IF   ( ?x flies )&( ?x lays eggs ))
         (THEN ( ?x is bird )))
(RULE 5  (IF   ( ?x is mammal )&( ?x eats meat ))
         (THEN ( ?x is carnivore )))
```

Forward Chaining Example

Facts:

- F1: Ungee gives milk
- F2: Ungee eats meat
- F3: Ungee has hoofs

Rules:

- R1: If X gives milk, then it is a mammal
- R2: If X is a mammal and eats meat, then carnivore.
- R3: If X is a carnivore and has hoofs, then ungulate

Easy to see: Ungee is ungulate.

Backward Chaining

The backward chaining mode of operation means that the systems begins with a goal and successively examines any rules with matching consequents. These candidate rules are considered one at a time.

The unmet conditions are in turn reintroduced as new goals. The control procedure then shifts attention recursively toward the new goal. The effort terminates when the top goal is finally satisfied.

Backward Chaining Algorithm

1. *Repeat*
2. *For* each hypothesis do
3. *For* each rule whose consequent matches this hypothesis
4. - try to *Match* each of the rule antecedents with the
5. facts from the working memory, or using backward chaining through another rule, thus, creating new hypotheses
6. - if all of the rule antecedents are supported then success
7. *until* all hypotheses have been tried, and none have been supported,
8. or until the goal is satisfied

Backward Chaining

Start with Goal G1: is Ungee an ungulate?

G1 matches conclusion of R3

Sets up premises as subgoals

- G2: Ungee is carnivore
- G3: Ungee has hoofs

G3 matches fact F3 so true.

G2 matches conclusion of R2. etc.

Example

Rules:

R1: IF hot AND smoky THEN fire

R2: IF alarm_beeps THEN smoky

R3: If fire THEN switch_on_sprinklers

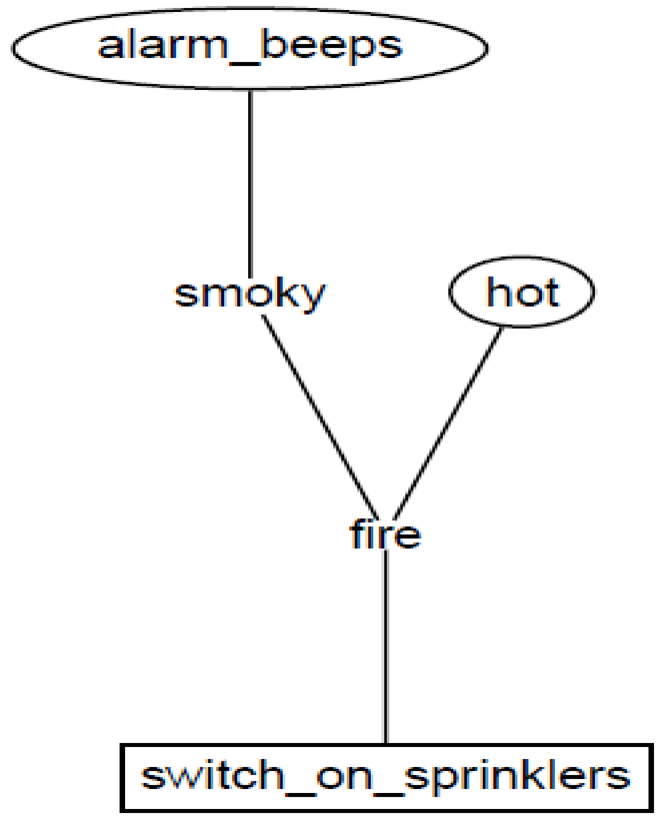
Facts:

F1: hot

F2: alarm_beeps

Goal:

Should I switch sprinklers on?



Rule Based Expert System

Knowledge is a theoretical or practical understanding of a subject or a domain. Knowledge is also the sum of what is currently known, and apparently knowledge is power. Those who possess knowledge are called experts.

Anyone can be considered a **domain expert** if he or she has deep knowledge (of both facts and rules) and strong practical experience in a particular domain. The area of the domain may be limited. In general, an expert is a skillful person who can do things other people cannot.

The human mental process is internal, and it is too complex to be represented as an algorithm.

However, most experts are capable of expressing their knowledge in the form of **rules** for problem solving.

```
IF the ‘traffic light’ is green
    THEN the action is go
IF the ‘traffic light’ is red
    THEN the action is stop
```

Rules as a knowledge representation technique

The term ***rule*** in AI, which is the most commonly used type of knowledge representation, can be defined as an IF-THEN structure that relates given information or facts in the IF part to some action in the THEN part. A rule provides some description of how to solve a problem. Rules are relatively easy to create and understand.

Any rule consists of two parts: the IF part, called

the ***antecedent*** (***premise*** or ***condition***) and the THEN part

called the ***consequent*** (***conclusion*** or ***action***).

IF <antecedent>

THEN <consequent>

A rule can have multiple antecedents joined by the keywords **AND** (**conjunction**), **OR** (**disjunction**) or a combination of both.

IF <antecedent 1> IF <antecedent 1>

AND <an t e **.**cedent 2> OR <a n t e**.**cedent 2> **.... ....**

AND <antecedent *n*> OR <antecedent *n*>

THEN <consequent> THEN <consequent>

The main players in the development team

There are five members of the expert system

development team:

the **domain expert**,

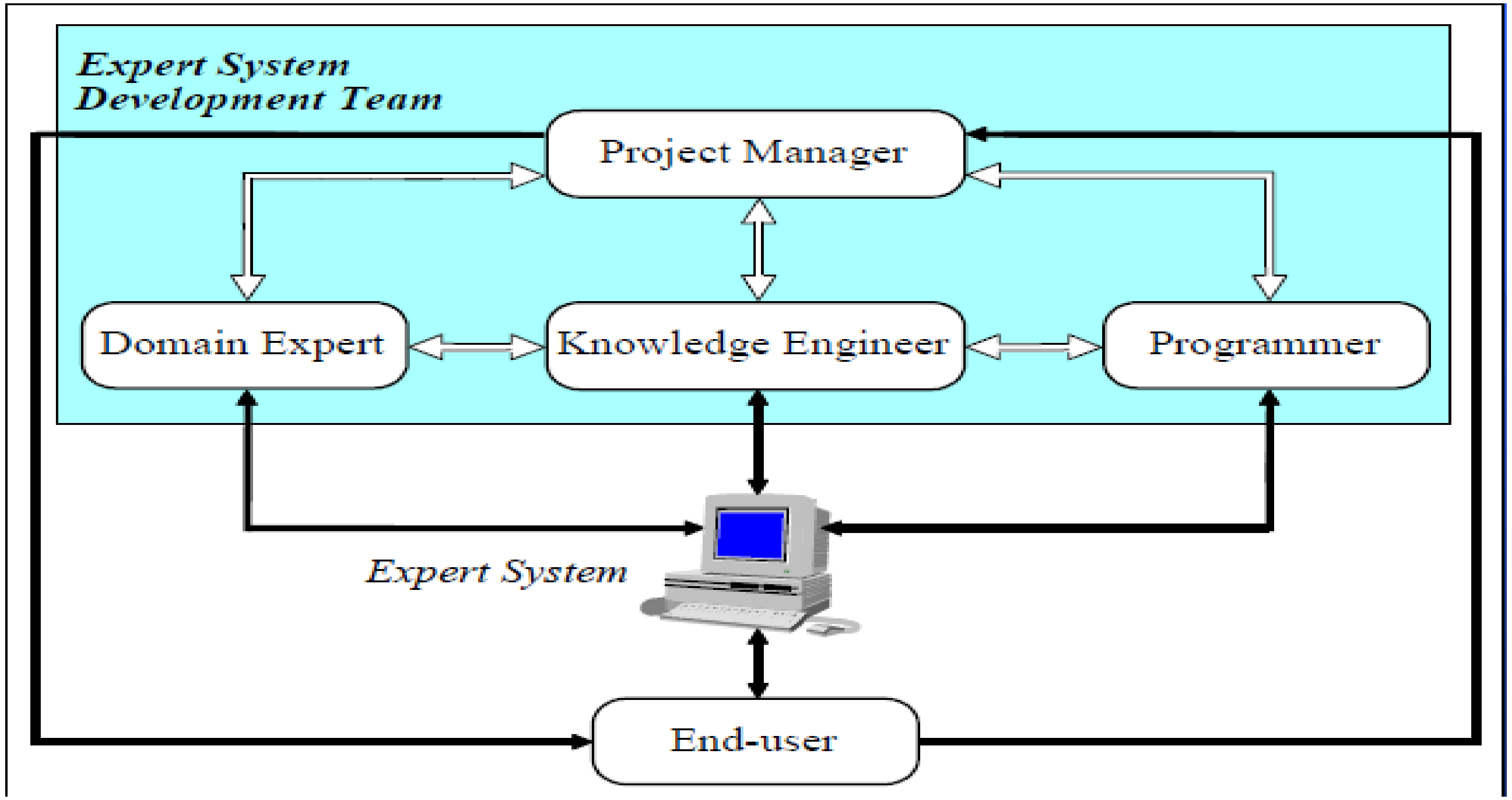
the **knowledge engineer**,

the **programmer**,

The **project manager** and the **end-user**.

The success of their expert system entirely depends on how well the members work together.

The main players in the development team



The ***domain expert*** is a knowledgeable and skilled person capable of solving problems in a specific area or ***domain***. This person has the greatest expertise in a given domain. This expertise is to be captured in the expert system. Therefore, the expert must be able to communicate his or her knowledge, be willing to participate in the expert system development and commit a substantial amount of time to the project. The domain expert is the most important player in the expert system development team.

The ***knowledge engineer*** is someone who is capable of designing, building and testing an expert system. He or she interviews the domain expert to find out how a particular problem is solved. The knowledge engineer establishes what reasoning methods the expert uses to handle facts and rules and decides how to represent them in the expert system. The knowledge engineer then chooses some development software or an expert system shell, or looks at programming languages for encoding the knowledge. And finally, the knowledge engineer is responsible for testing, revising and integrating the expert system into the workplace.

The ***programmer*** is the person responsible for the actual programming, describing the domain knowledge in terms that a computer can understand. The programmer needs to have skills in symbolic programming in such AI languages as LISP, Prolog and OPS5 and also some experience in the application of different types of expert system shells. In addition, the programmer should know conventional programming languages like C, Pascal, FORTRAN and Basic.

The ***project manager*** is the leader of the expert system development team, responsible for keeping the project on track. He or she makes sure that all deliverables and milestones are met, interacts with the expert, knowledge engineer, programmer and end-user.

The ***end-user***, often called just the *user*, is a person who uses the expert system when it is developed. The user must not only be confident in the expert system performance but also feel comfortable using it. Therefore, the design of the user interface of the expert system is also vital for the project’s success;

the end-user’s contribution here can be crucial.

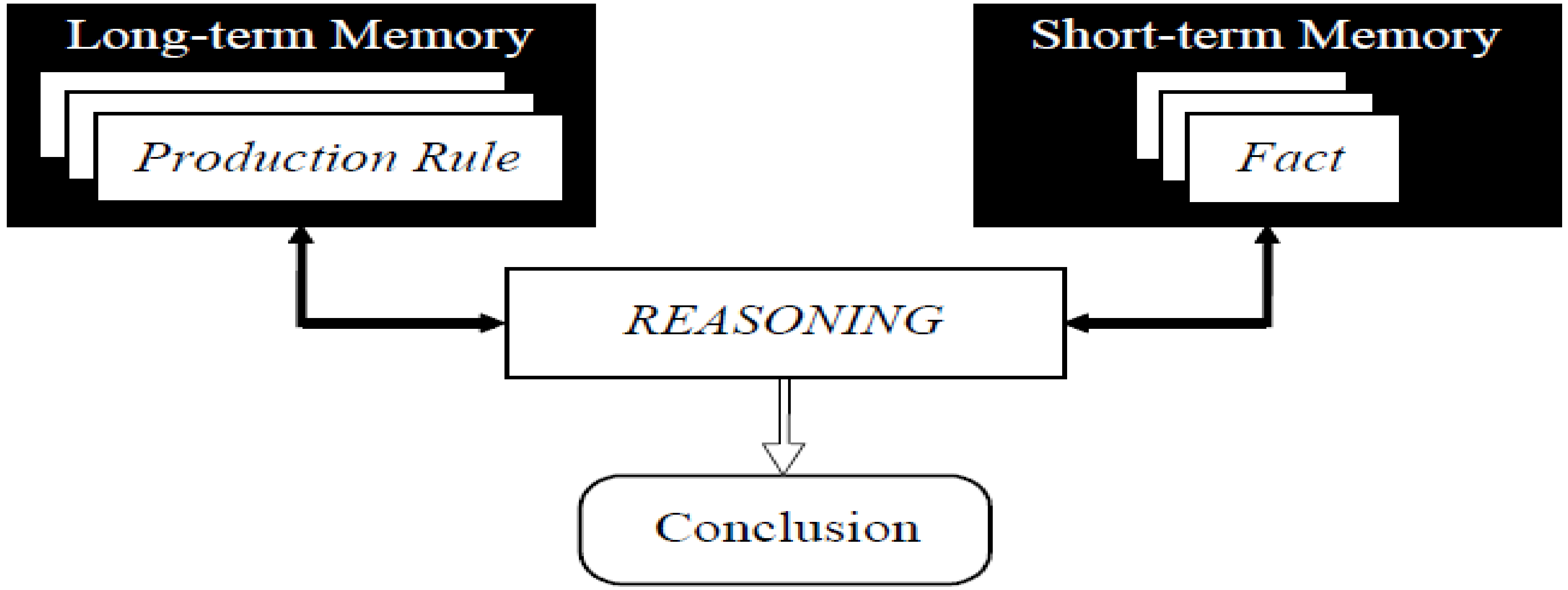
Structure of a rule-based expert system

In the early seventies, Newell and Simon from Carnegie-Mellon University proposed a production system model, the foundation of the modern rulebased expert systems.

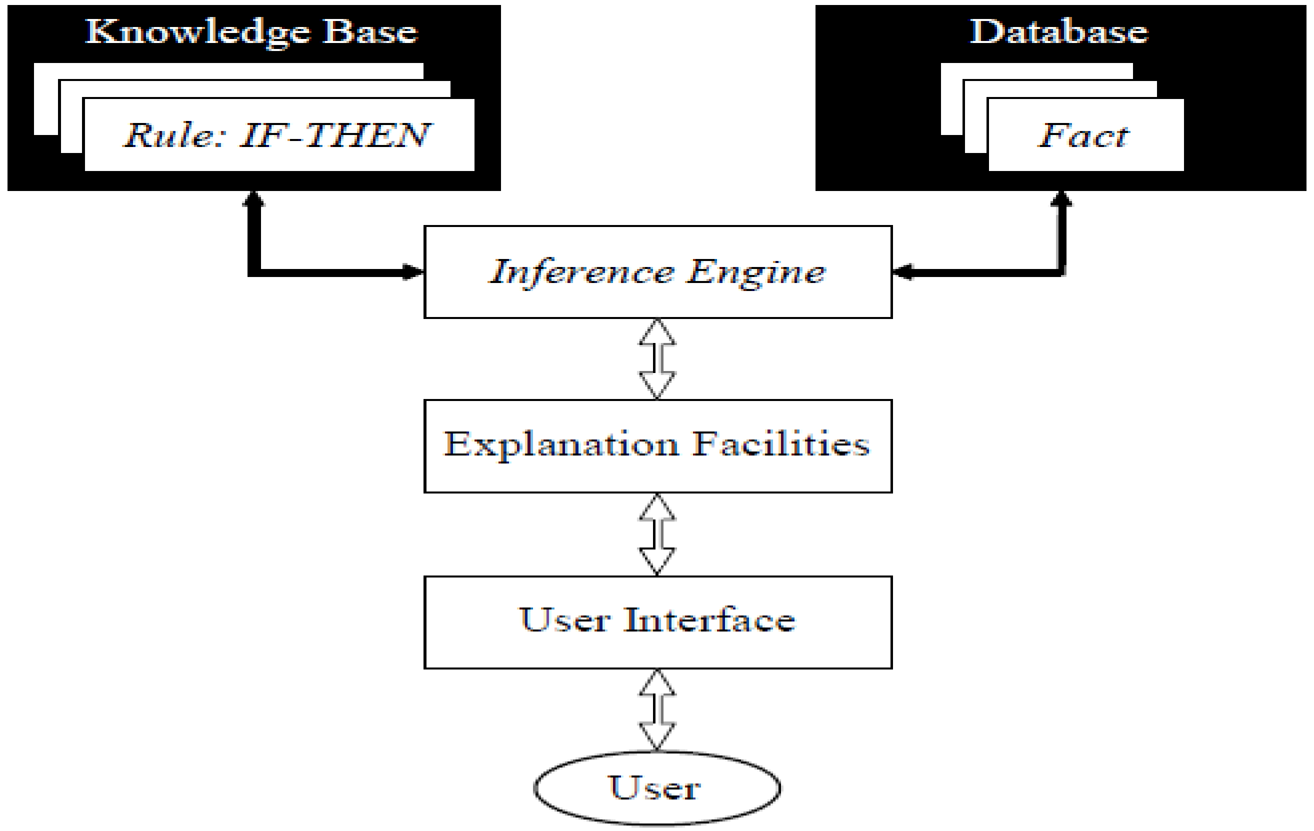
The production model is based on the idea that humans solve problems by applying their knowledge (expressed as production rules) to a given problem represented by problem-specific information.

The production rules are stored in the long-term memory and the problem-specific information or facts in the short-term memory.

Production system model



Basic structure of a rule-based expert system



The **knowledge base** contains the domain knowledge useful for problem solving. In a rule based expert system, the knowledge is represented as a set of rules. Each rule specifies a relation, recommendation, directive, strategy or heuristic and has the IF (condition) THEN (action) structure. When the condition part of a rule is satisfied, the rule is said to ***fire*** and the action part is executed.

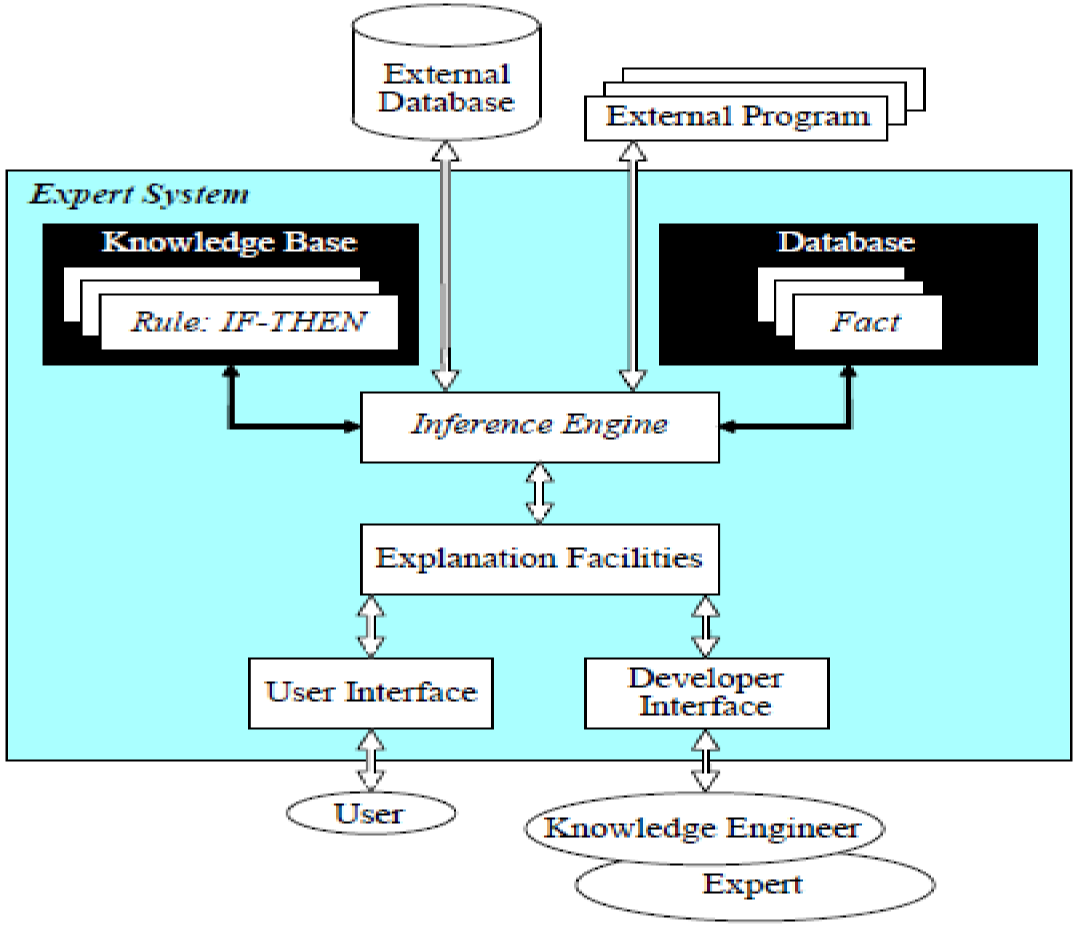
The **database** includes a set of facts used to match against the IF (condition) parts of rules stored in the knowledge base.

The **inference engine** carries out the reasoning whereby the expert system reaches a solution. It links the rules given in the knowledge base with the facts provided in the database.

The **explanation facilities** enable the user to ask the expert system ***how*** a particular conclusion is reached and ***why*** a specific fact is needed. An expert system must be able to explain its reasoning and justify its advice, analysis or conclusion.

The **user interface** is the means of communication between a user seeking a solution to the problem and an expert system.

Complete structure of a rule-based expert system



Characteristics of an expert system

An expert system is built to perform at a human expert level in a ***narrow, specialized domain***. Thus, the most important characteristic of an expert system is its high-quality performance. No matter how fast the system can solve a problem, the user will not be satisfied if the result is wrong.

On the other hand, the speed of reaching a solution is very important. Even the most accurate decision or diagnosis may not be useful if it is too late to apply, for instance, in an emergency, when a patient dies or a nuclear power plant explodes.

Expert systems apply **heuristics** to guide the reasoning and thus reduce the search area for a solution.

A unique feature of an expert system is its **explanation capability**. It enables the expert system to review its own reasoning and explain its decisions.

Expert systems employ **symbolic reasoning** when solving a problem. Symbols are used to represent different types of knowledge such as facts, concepts and rules.

Components of an ES

1. Knowledge Base
2. Reasoning or Inference Engine
3. User Interface
4. Explanation Facility

User Interface

- Allows the expert system and the user to communicate.
- Finds out what it is that the system needs to answer.
- Sends the user questions or answers and receives their response.

Explanation Facility

- Explains the systems reasoning and justifies its conclusions.

Knowledge Base

- Represents all the data and information imputed by experts in the field.

- Stores the data as a set of rules that the system must follow to make decisions.

Inference Engine

- Asks the user questions about what they are looking for.
- Applies the knowledge and the rules held in the knowledge base.
- Appropriately uses this information to arrive at a decision.

Advantages of rule-based expert systems

Natural knowledge representation. An expert usually explains the problem-solving procedure with such expressions as this: “In such-and-such situation, I do so-and-so”. These expressions can be represented quite naturally as IF-THEN production rules.

Uniform structure. Production rules have the uniform IF-THEN structure. Each rule is an independent piece of knowledge. The very syntax of production rules enables them to be self documented.

Separation of knowledge from its processing.

The structure of a rule-based expert system provides an effective separation of the knowledge base from the inference engine. This makes it possible to develop different applications using the same expert system shell.

Dealing with incomplete and uncertain knowledge.

Most rule-based expert systems are capable of representing and reasoning with incomplete and uncertain knowledge

Disadvantages of rule-based expert systems

Opaque relations between rules. Although the individual production rules are relatively simple and self-documented, their logical interactions within the large set of rules may be opaque. Rule-based systems make it difficult to observe how individual rules serve the overall strategy.

Ineffective search strategy. The inference engine applies an exhaustive search through all the production rules during each cycle. Expert systems with a large set of rules (over 100 rules) can be slow, and thus large rule-based systems can be unsuitable for real-time applications.

Inability to learn. In general, rule-based expert systems do not have an ability to learn from the experience. Unlike a human expert, who knows when to “break the rules”, an expert system cannot automatically modify its knowledge base, or adjust existing rules or add new ones. The knowledge engineer is still responsible for revising and maintaining the system.

MYCIN

History and Overview

MYCIN Architecture

Consultation System

- Knowledge Representation & Reasoning

Explanation System

Knowledge Acquisition

Results, Conclusions

History

Thesis Project by Shortliffe @ Stanford

Davis, Buchanan, van Melle, and others
- Stanford Heuristic Programming Project
- Infectious Disease Group, Stanford Medical

Project Spans a Decade
- Research started in 1972
- Original implementation completed 1976
- Research continues into the 80’s

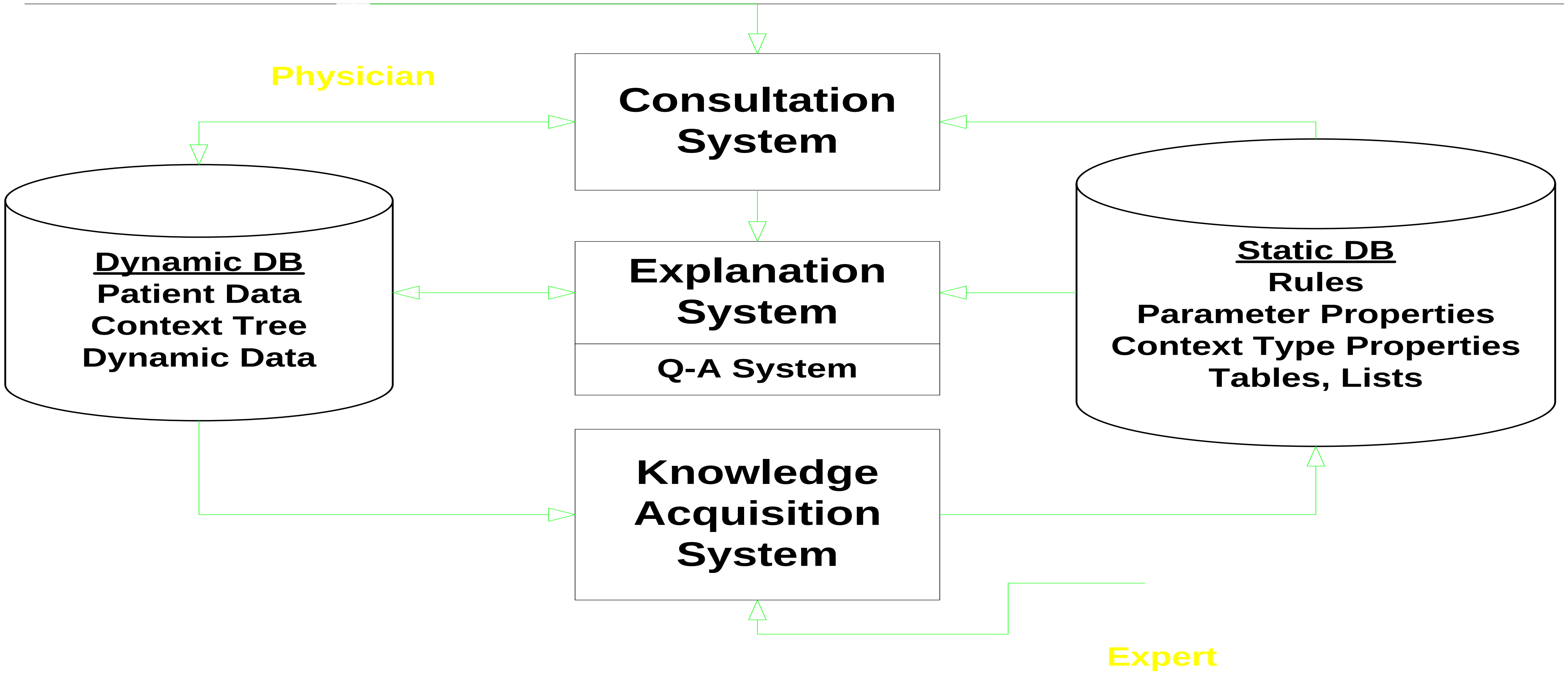
Tasks and Domain

Disease DIAGNOSIS and Therapy SELECTION

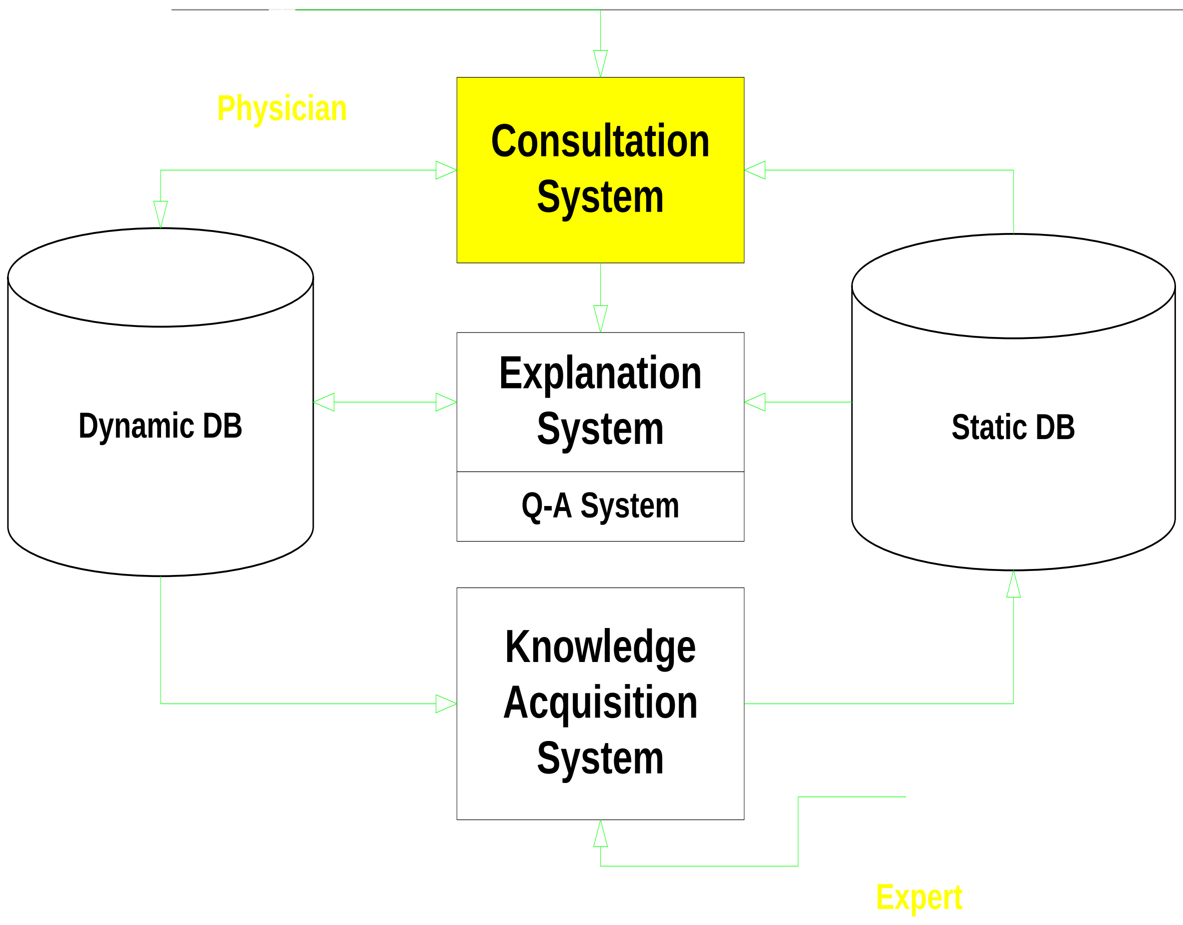
Advice for non-expert physicians with time considerations and <u>in</u>complete evidence on:

- Bacterial infections of the blood
- Expanded to meningitis and other ailments

MYCIN Architecture



Consultation System



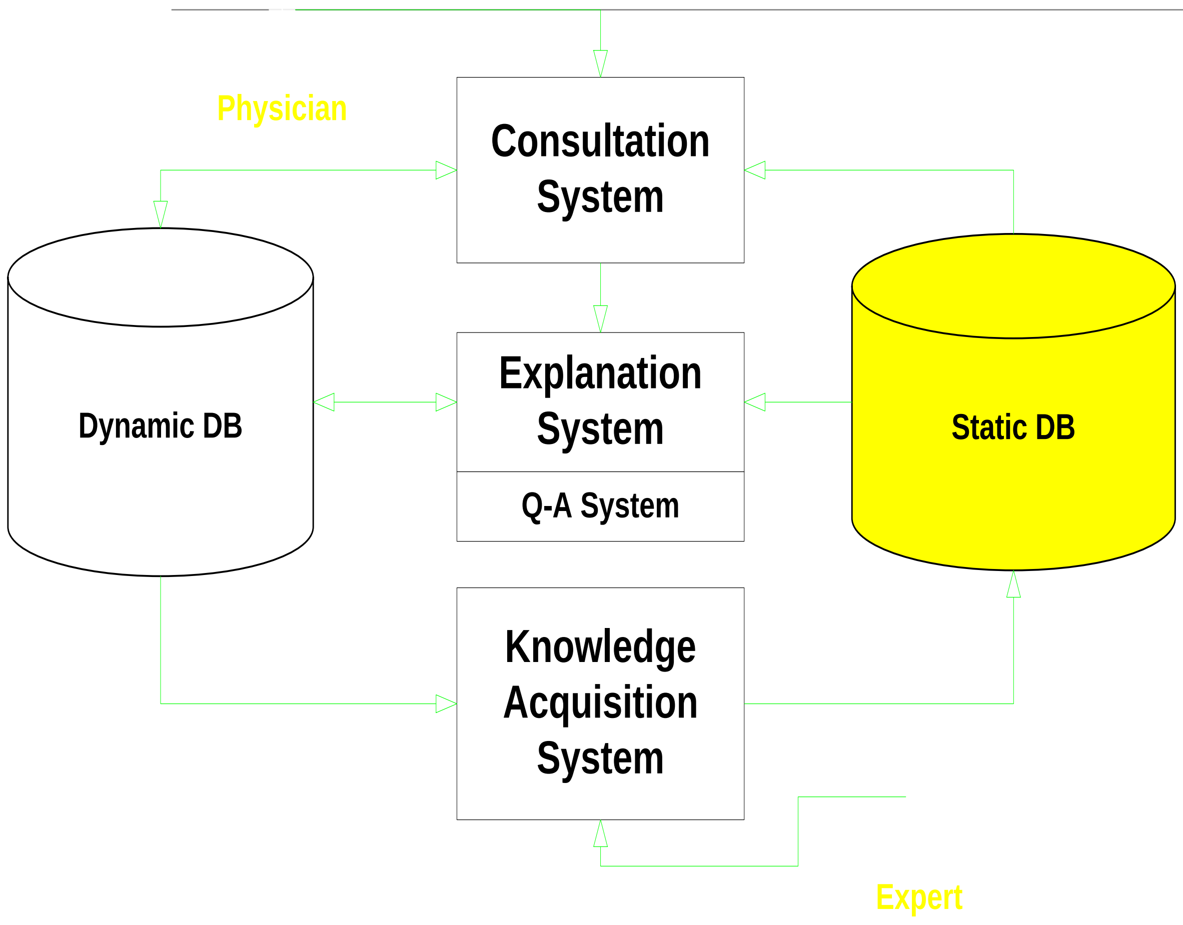
Performs Diagnosis and Therapy Selection

Control Structure reads Static DB (rules) and read/writes to Dynamic DB (patient, context)

Linked to Explanations

Terminal interface to Physician

Static Database



Rules

Meta-Rules

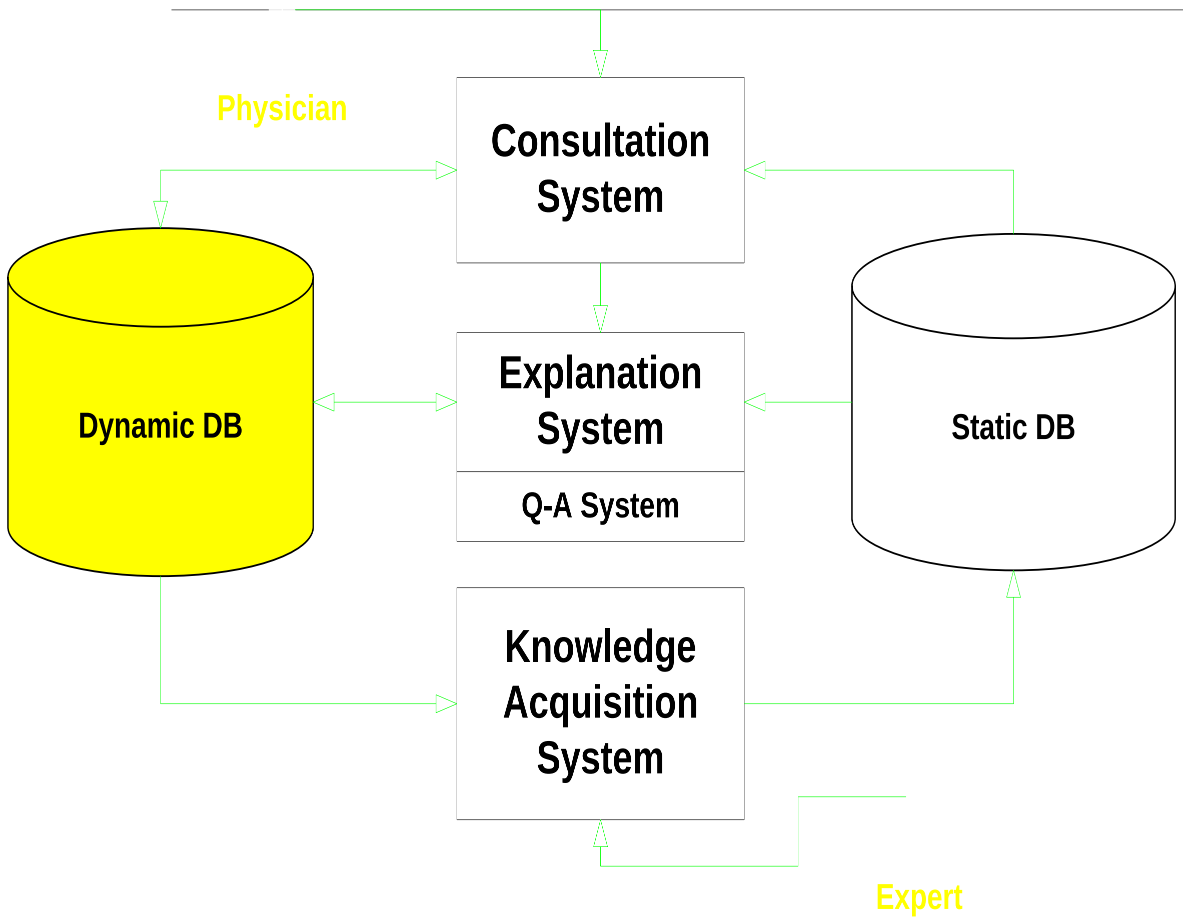
Templates

Rule Properties

Context Properties

Fed from Knowledge Acquisition System

Dynamic Database



Patient Data

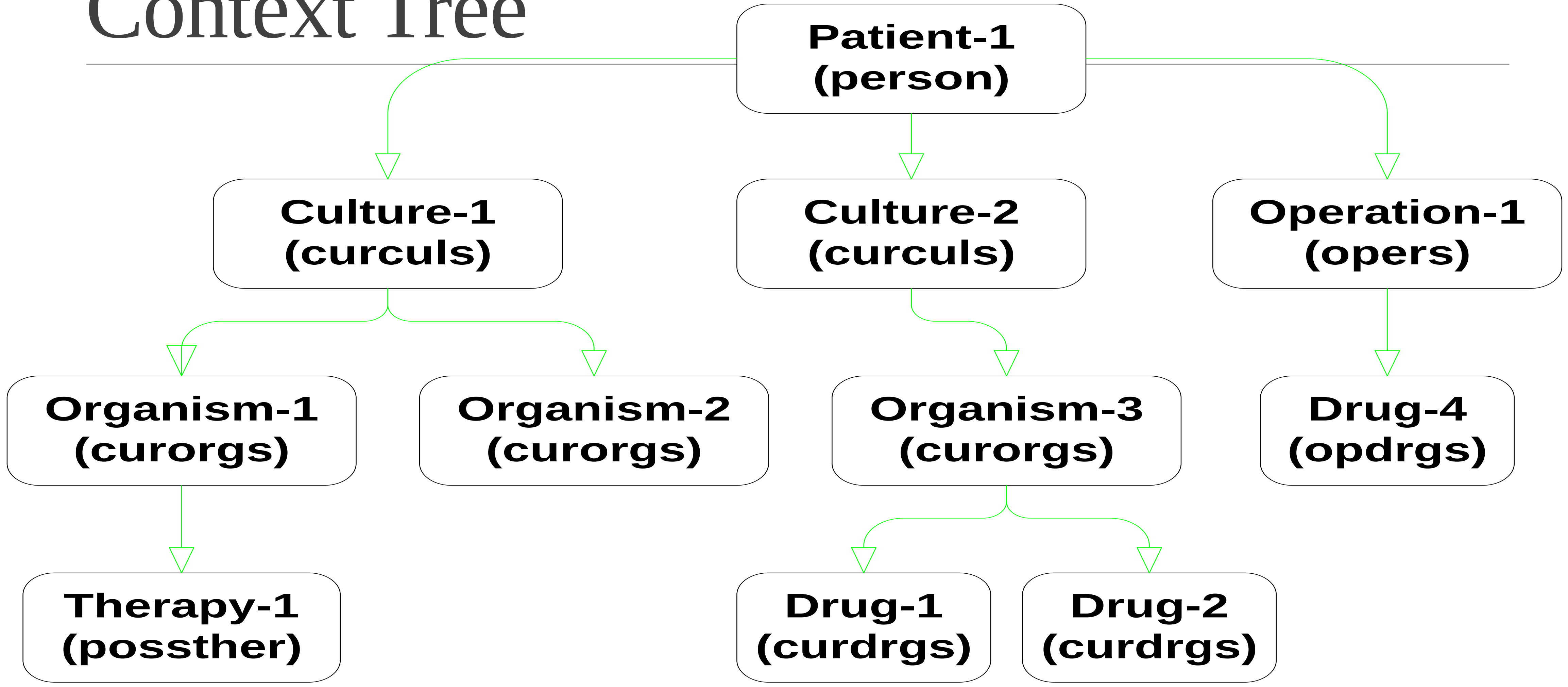
Laboratory Data

Context Tree

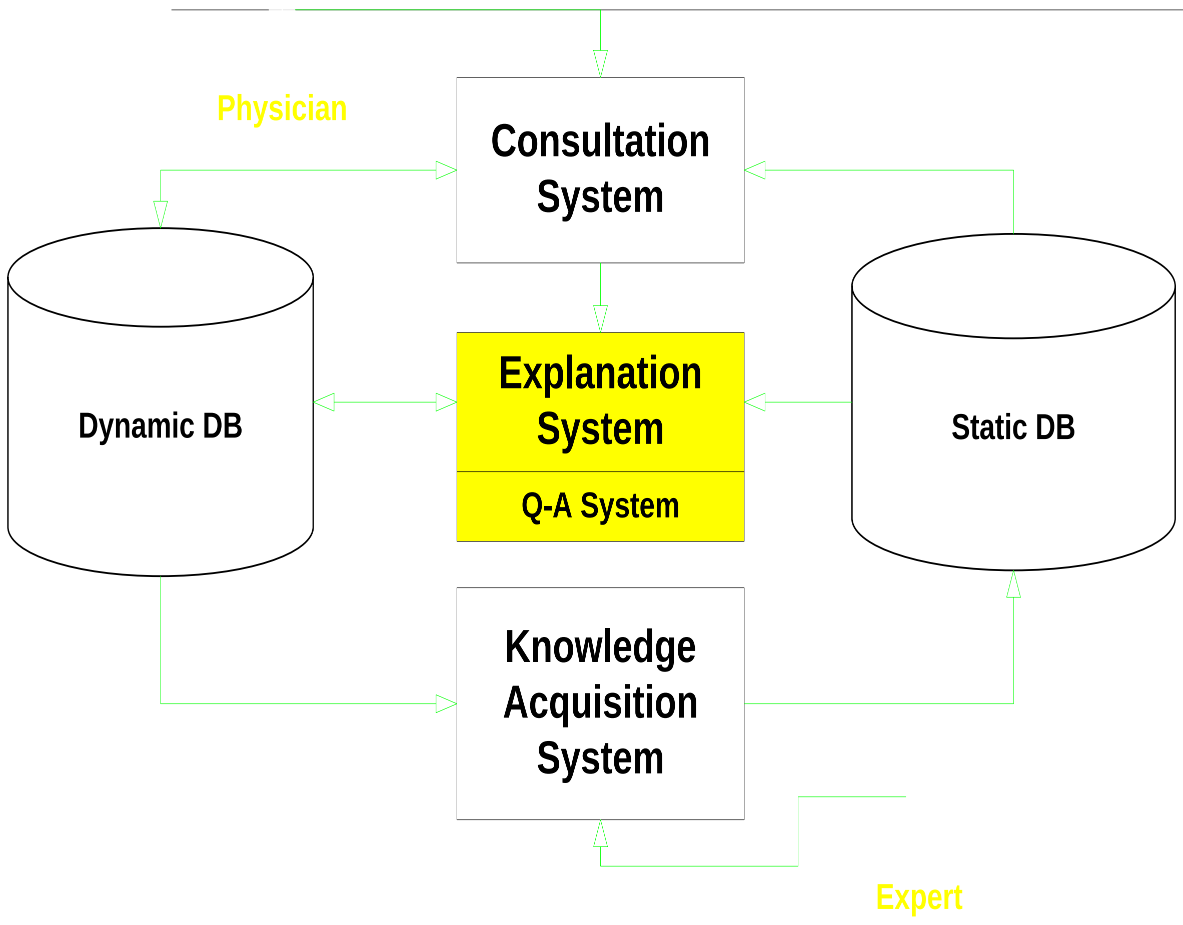
Built by Consultation System

Used by Explanation System

Context Tree



Explanation System

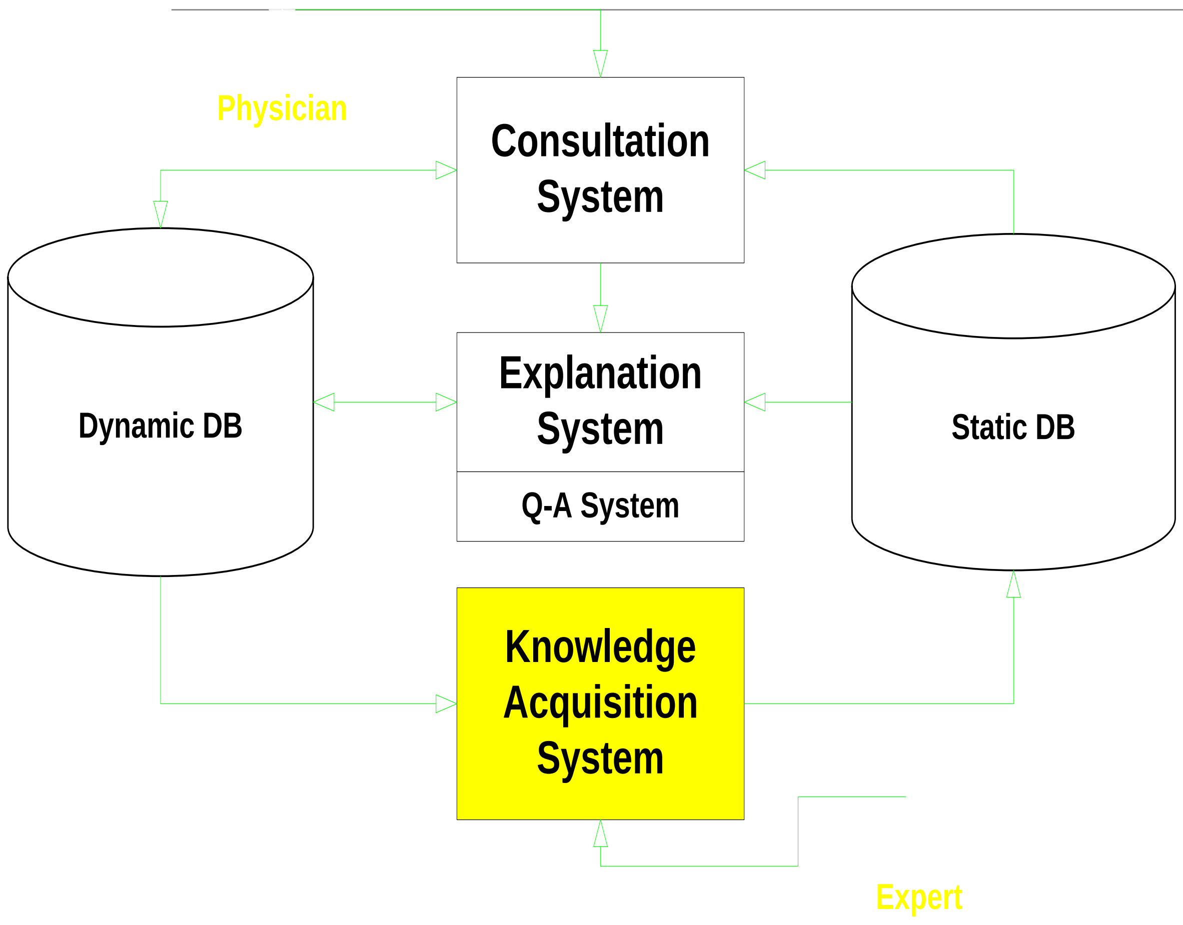


Provides reasoning why a conclusion has been made, or why a question is being asked

Q-A Module

Reasoning Status Checker

Knowledge Acquisition System



Extends Static DB via Dialogue with Experts

Dialogue Driven by System

Requires minimal training for Experts

Allows for Incremental Competence, NOT an All-or-Nothing model

Knowledge Acquisition

IF-THEN Symbolic logic was found to be easy for experts to learn, and required little training by the MYCIN team

When faced with a rule, the expert must either except it or be forced to update it using the education process

Education Process

1. Bug is uncovered, usually by Explanation process
2. Add/Modify rules using *subset of English* by experts
3. Integrating new knowledge into KB
   - Found to be difficult in practice, requires detection of contradictions, and complex concepts become difficult to express