Artificial Intelligence

Unit 5 Expert System

What is an expert system?

- “An expert system is a computer system that emulates, or acts in all respects, with the decision-making capabilities of a human expert.”
- Expert Systems = knowledge-based systems = knowledge-based expert systems

Expert System

- Expert System is one of the subareas of AI. An expert system is a set of programs that manipulate knowledge to solve problem in a specialized area
- An expert system is more than a database program. A database program retrieve and store the data

What is an expert system?

- The basic idea is that if a human expert can specify the steps of reasoning by which a problem may be solved, so too can an expert system.
- Restricted domain expert systems (extensive use of specialized knowledge at the level of human expert) function well which is not the case of general-purpose problem solver.

Prominent Expert Systems

- MYCIN – used to diagnose infectious blood diseases and recommend antibiotics.
- DENDRAL – embedded a chemist's knowledge of mass spectrometry rules to use in analysis.
- CADUCEUS – used to analyze blood-borne infectious bacteria
- CLIPS and Prolog programming languages are both used in expert systems
  - The Age of Empire game uses CLIPS to control its AI

Benefits of Expert Systems

- An ES is no substitute for a knowledge worker's overall performance of the problem-solving task. But these systems can dramatically reduce the amount of work the individual must do to solve a problem, and they do leave people with the creative and innovative aspects of problem solving.
- Some of the possible organizational benefits of expert systems are:

1. An Es can complete its part of the tasks much faster than a human expert.
2. The error rate of successful systems is low, sometimes much lower than the human error rate for the same task.
3. ESs make consistent recommendations
4. ESs are a convenient vehicle for bringing to the point of application difficult-to-use sources of knowledge.
5. ESs can capture the scarce expertise of a uniquely qualified expert.
6. ESs can become a vehicle for building up organizational knowledge, as opposed to the knowledge of individuals in the organization.
7. When use as training vehicles, ESs result in a faster learning curve for novices.
8. The company can operate an ES in environments hazardous for humans.

Limitations of Expert Systems(ESs)

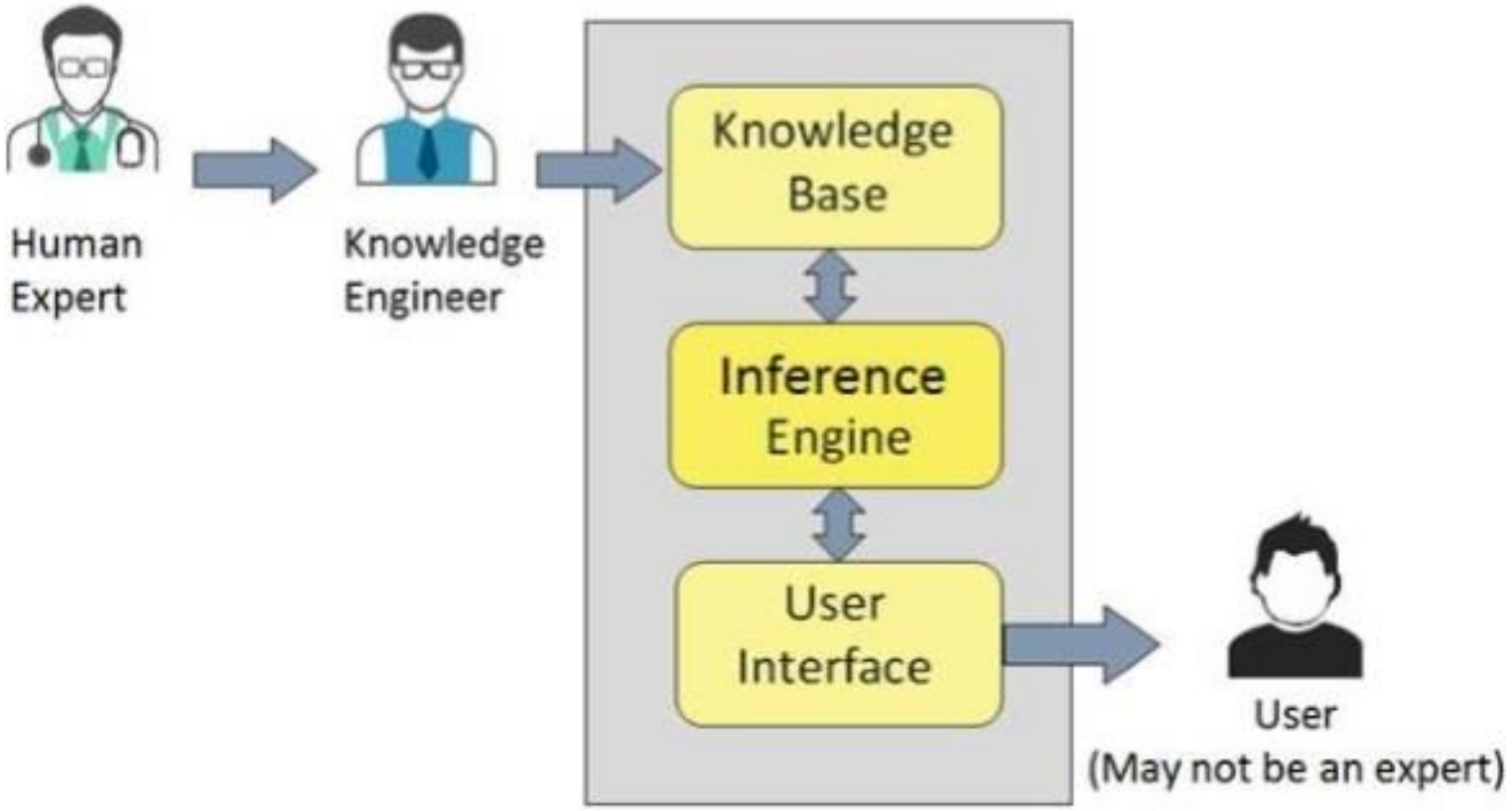
- No technology offers an easy and total solution. Large systems are costly and require significant development time and computer resources. ESs also have their limitations which include:

1. Limitations of the technology
2. Problems with knowledge acquisition
3. Operational domains as the principal area of ES application
4. Maintaining human expertise in organizations

Expert System Main Components

- Knowledge base – obtainable from books, magazines, knowledgeable persons, etc.; or expertise knowledge. Knowledge base of an expert system contains both behavioral and procedural knowledge.
- Inference engine – draws conclusions from the knowledge base.

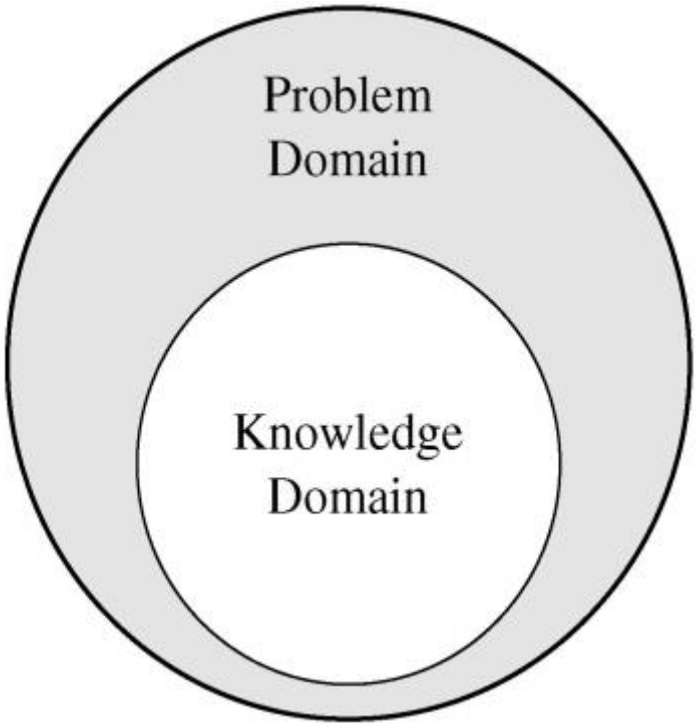
Basic Expert System Architecture



Problem Domain vs. Knowledge Domain

- In general, the first step in solving any problem is defining the problem area or domain to be solved.
- An expert's knowledge is specific to one problem domain – medicine, finance, science, engineering, etc.
- The expert's knowledge about solving specific problems is called the knowledge domain.
- The problem domain is always a superset of the knowledge domain.
- Expert system reasons from knowledge domain.
- **Example:** infections diseases diagnostic system does not have (or require) knowledge about other branches such as surgery.

Problem and Knowledge Domain Relationship



Advantages of Expert Systems

- Increased availability: on suitable computer hardware
- Reduced cost
- Reduced danger: can be used in suitable environment.
- Permanence: last for ever, unlike human who may die, retire, quit.
- Multiple expertise:
- Increased reliability

Advantages of Expert Systems

- Explanation: explain in detail how arrived at conclusions.
- Fast response: (e.g. emergency situations).
- Steady, unemotional, and complete responses at all times: unlike human who may be inefficient because of stress or fatigue.
- Intelligent tutor: provides direct instructions (student may run simple programs and explaining the system's reasoning).
- Intelligent database: access a database intelligently (e.g. data mining).

Building an Expert System

- Can be built from scratch – using lots of if-then-else statements.
- There are many products being sold to make programming these systems easier.
  - A few that I'm aware of:
    - Exsys – www.exsys.com – provides an easy to use user interface to develop traditional applications or web-based solutions.
    - Barisoft – www.barisoft.com – developed by a PSU professor and finely tuned by his students
    - CLIPS - provides a complete environment for the construction of rules and/or object based expert systems
    - Jess – the Rule Engine, built on top of CLIPS, for the Java Platform

Expert System Building Tools

- Shells and skeletons
- When system can be built without any domain specific knowledge. Such systems are known as skeletons systems, shells or simply AI tools.
- A shell often includes tools that help with the design, development and testing of the knowledge base.
- With the shell approach expert system representing many different problem domains may be developed and delivered with the same software environment.

Shells (Tools)

- Many commercial shells are available today, ranging in size from shells on PCs, to shells on workstations. Its range from hundred to tens of thousand dollars, and range in complexity from simple.
- Some of such tools/shells and their applications are given below

Shell tools

- ART(Automated Reasoning tools)
- ADS (Aion Development System)- os
- Acquire – Step by step methodology used for knowledge base system .
- Babylon- is modular configurable (is used to logical programming)
- Expert ease – designed for decision support which can run on microcomputer such as IBM PC.

Shell tools

- GURU – is used DBMS and RDBMS
- HUGIN – is used for probability function
- KEE (Knowledge Engineering Environment) – is used for frames and scripts.
- KES (Knowledge Engineering System) – Available in different versions to run on machines.

Expert System Architecture

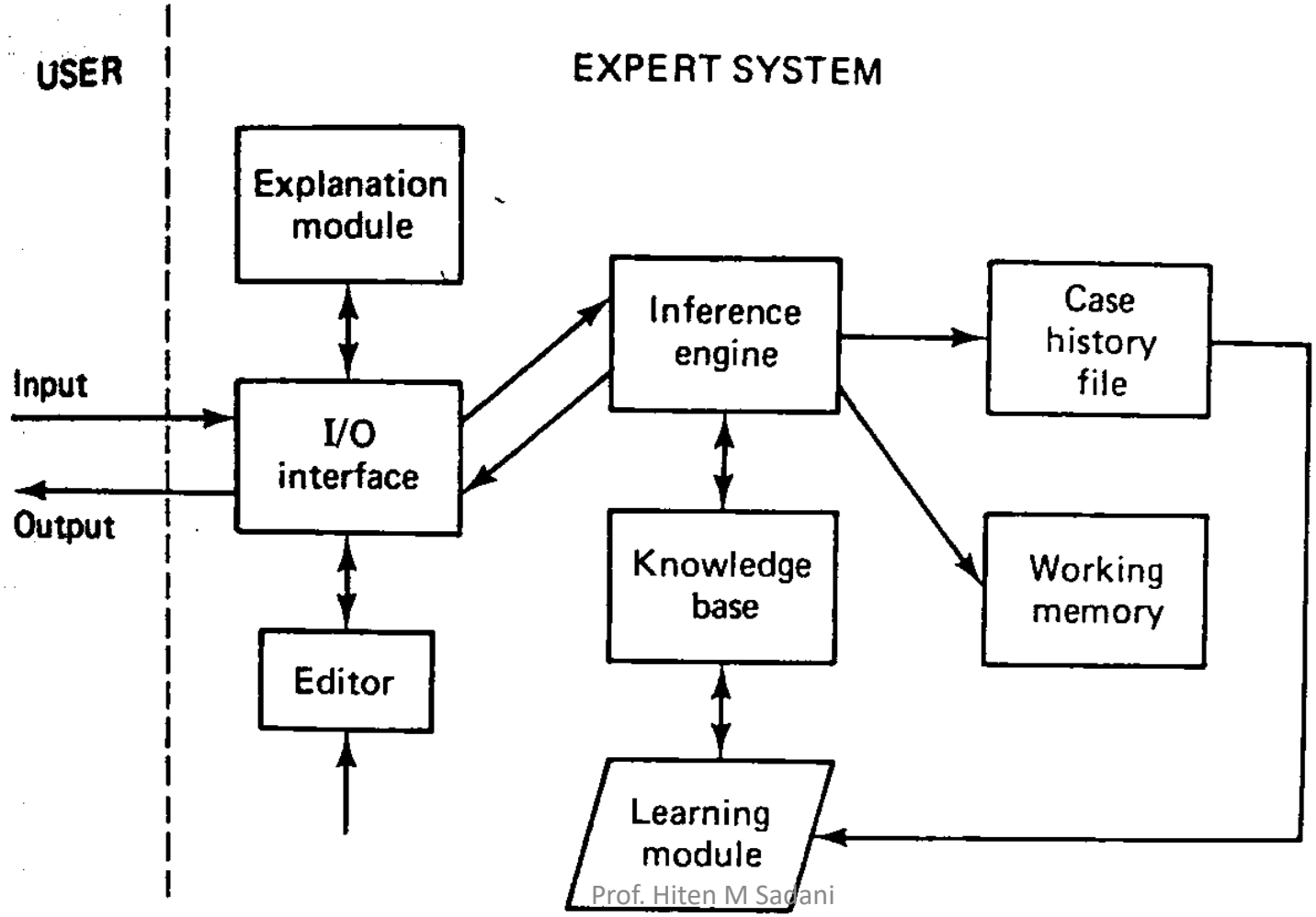
- There are many architectures, on which expert systems are built upon.

1. Ruled based system architecture - is used in expert and other types of knowledge based system in the production systems also called as rule base system.
2. Non production system architecture

Ruled based system architecture

- Rule bases system is same as the production system.
- When we creating and maintain the production system and find initial state to goal state then rule base system is used.
- Learning module and history file are not common components of expert systems when they are provided they are used to assist in building and refining the knowledge base.

Structure of a Rule-Based Expert System architecture



How Expert Systems Work

- The strength of an ES derives from its ***knowledge base*** - an organized collection of facts and heuristics about the system's domain.
- An ES is built in a process known as ***knowledge engineering***, during which knowledge about the domain is acquired from human experts and other sources by knowledge engineers.
- The accumulation of knowledge in knowledge bases, from which conclusions are to be drawn by the inference engine, is the hallmark of an expert system.

User Interface

- User interface provides interaction between user of the ES and the ES itself. It is generally Natural Language Processing so as to be used by the user who is well-versed in the task domain. The user of the ES need not be necessarily an expert in Artificial Intelligence.
- It explains how the ES has arrived at a particular recommendation. The explanation may appear in the following forms −
  - Natural language displayed on screen.
  - Verbal narrations in natural language.
  - Listing of rule numbers displayed on the screen.

Requirements of Efficient ES User Interface

- The user interface makes it easy to trace the credibility of the deductions.
- It should help users to accomplish their goals in shortest possible way.
- It should be designed to work for user's existing or desired work practices.
- Its technology should be adaptable to user's requirements; not the other way round.
- It should make efficient use of user input.

Knowledge Representation and the Knowledge Base

- The knowledge base of an ES contains both factual and heuristic knowledge.
- ***Knowledge representation*** is the method used to organize the knowledge in the knowledge base.
- Knowledge bases must represent notions as actions to be taken under circumstances, causality, time, dependencies, goals, and other higher-level concepts.

Methods of knowledge representation

- Propositional Logic
- Predicate Logic or FOL(First Order Predicate Logic)
- Semantic Net
- Scripts
- Frame
- Conceptual Dependency Network
- Production Rules
- etc…

Production Rules

- Production rules are the most common method of knowledge representation used in business.
- Rule-based expert systems are expert systems in which the knowledge is represented by production rules.
- A production rule, or simply a rule, consists of an IF part (a condition or premise) and a THEN part (an action or conclusion). IF condition THEN action (conclusion).

Rule Based Production Systems

- Production system - uses knowledge in the form of rules to provide diagnoses or advice on the basis of input data.
- Parts
  - Database of rules (knowledge base)
  - Database of facts
  - Inference engine which reasons about the facts using the rules

Production system

- A **production system** is a model of computation that provides pattern-directed search control using a set of **production rules**, a **working memory**, and a **recognize-act cycle**.

- The **productions** are rules of the form **C → A**, where the LHS is known as the **condition** and the RHS is known as the **action**. These rules are interpreted as follows: *given condition, C, take action A*. The action part can be any step in the problem solving process. The condition is the pattern that determines whether the rule applies or not.

Production system

- **Working memory** contains a description of the **current state of the world** in the problem-solving process. The description is matched against the conditions of the production rules. When a conditions matches, its action is performed. Actions are designed to alter the contents of working memory.

- The **recognize-act cycle** is the control structure. The patterns contained in working memory are matched against the conditions of the production rules, which produces a subset of rules known as the **conflict set**, whose conditions match the contents of working memory. One (or more) of the rules in the conflict set is selected (**conflict resolution**) and **fired**, which means its action is performed. The process terminates when no rules match the contents of working memory.

Conflict Resolution

- Problem: more than one rule fires at once
- Conflict resolution strategy decides which conclusions to use.
  - Give rules priorities and to use the conclusion that has the highest priority.
  - Apply the rule that was most recently added to the database.

Example

String Rewriting

- Suppose we have the following set of productions:
  - 1. ba → ab
  - 2. ca → ac
  - 3. cb → bc
- A production *matches* if its LHS matches any portion of the string in working memory. The **conflict resolution rule** in this example is to choose the lowest numbered rule.

Explanation Module

- The ***explanation facility*** explains how the system arrived at the recommendation.
- Depending on the tool used to implement the expert system, the explanation may be either in a natural language or simply a listing of rule numbers.

Inference Engine

- The inference engine:
- 1. Combines the facts of a specific case with the knowledge contained in the knowledge base to come up with a recommendation. In a rule-based expert system, the inference engine controls the order in which production rules are applied and resolves conflicts if more than one rule is applicable at a given time. This is what are reasoning amounts to in rule-based systems.
- 2. Directs the user interface to query the user for any information it needs for further inferencing.
- The facts of the given case are entered into the **working memory**, which acts as a blackboard, accumulating the knowledge about the case at hand. The inference engine repeatedly applies the rules to the working memory, adding new information (obtained from the rules conclusions) to it, until a goal state is produced or confirmed.

Strategies of Inference Engine

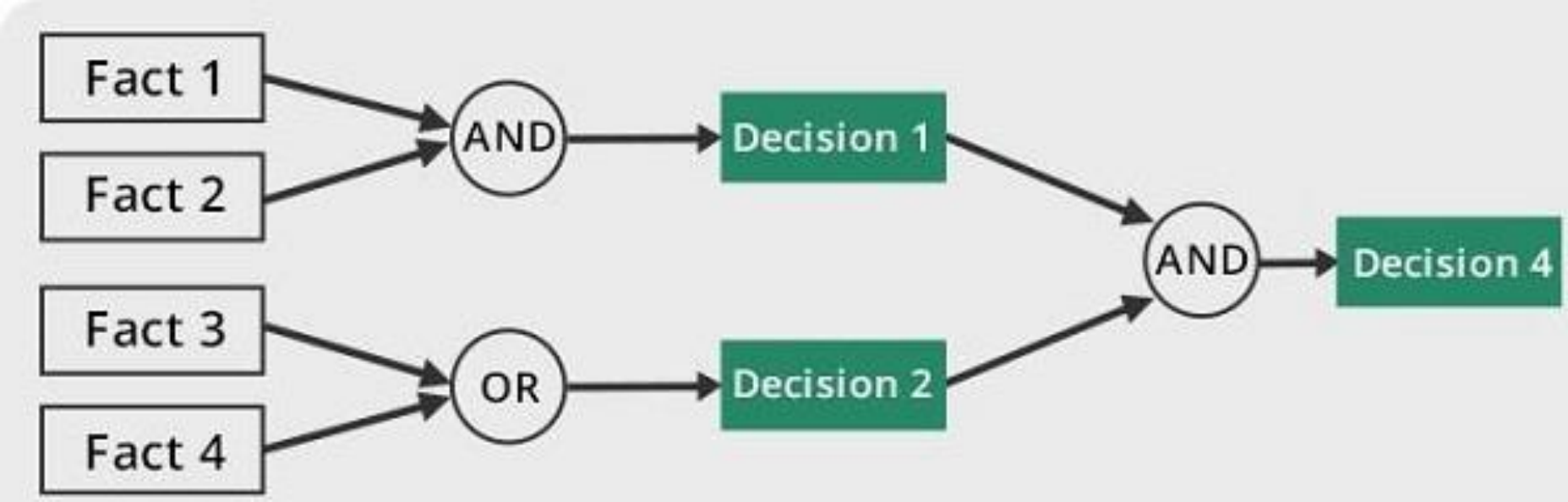
- Inferencing engines for rule-based systems generally work by either forward or backward chaining of rules. Two strategies are:

1. ***Forward chaining***
2. ***Backward chaining***

Forward chaining

- Forward chaining is a data-driven strategy.
- The inferencing process moves from the facts of the case to a goal (conclusion). The strategy is thus driven by the facts available in the working memory and by the premises that can be satisfied.
- The inference engine attempts to match the condition (IF) part of each rule in the knowledge base with the facts currently available in the working memory. If several rules match, a conflict resolution procedure is invoked; for example, the lowest-numbered rule that adds new information to the working memory is fired. The conclusion of the firing rule is added to the working memory.
- Forward-chaining systems are commonly used to solve more open-ended problems of a design or planning nature, such as, for example, establishing the configuration of a complex product.

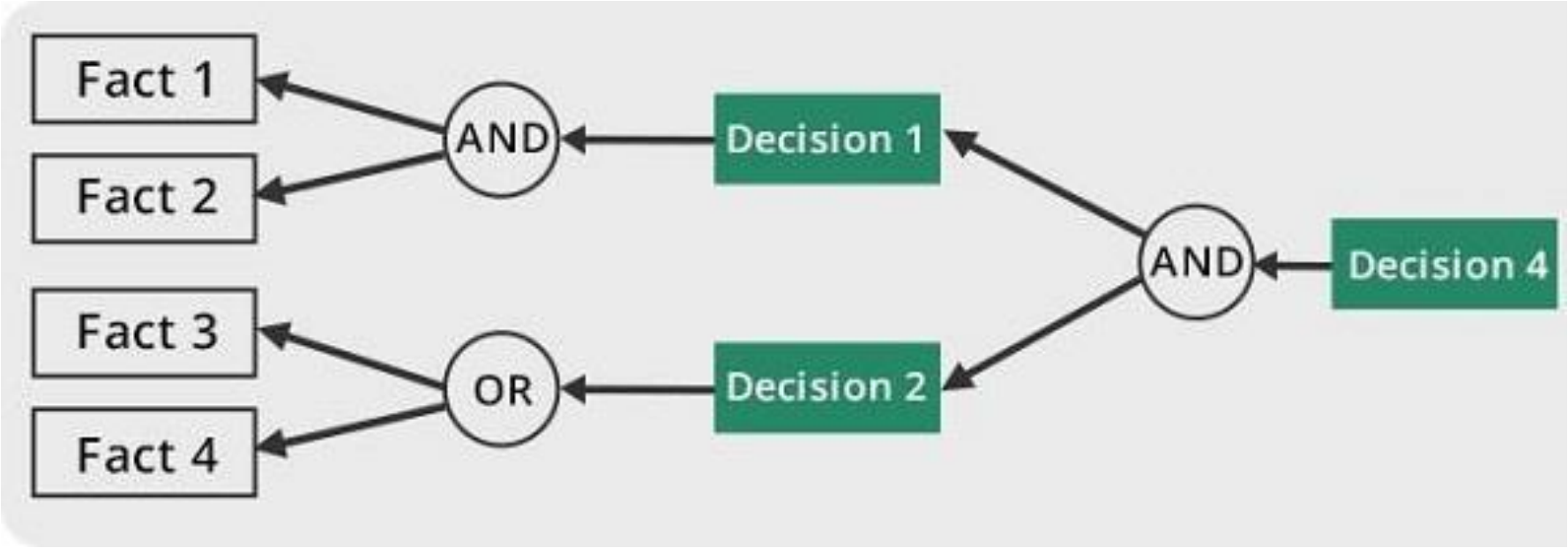
Forward chaining



Backward chaining

- Backward chaining is the inference engine attempts to match the assumed (hypothesized) conclusion - the goal or sub goal state - with the conclusion (THEN) part of the rule. If such a rule is found, its premise becomes the new sub goal. In an ES with few possible goal states, this is a good strategy to pursue.
- If a hypothesized goal state cannot be supported by the premises, the system will attempt to prove another goal state. Thus, possible conclusions are review until a goal state that can be supported by the premises is encountered.
- Backward chaining is best suited for applications in which the possible conclusions are limited in number and well defined. Classification or diagnosis type systems, in which each of several possible conclusions can be checked to see if it is supported by the data, are typical applications.

Backward chaining



Inference Process

- The inference engine accepts user input queries and responses to questions through the I/O interface and uses this dynamic information together with the static knowledge (the rules and facts) stored in the knowledge base.
- The knowledge in the knowledge base is used to derive conclusions about the current case or situation as presented by the user's input.
- The inferring process is carried out recursively in three stages:
  - Match
  - Select
  - Execute

Reasoning with production rules

- The statements forming the conditions, or the conclusions, in such rules, may be structures, following some syntactic convention (such as three items enclosed in brackets).

Reasoning with production rules

- Very often, these structures will include variables - such variables can, of course, be given a particular value, and variables with the same name in the same rule will share the same value.

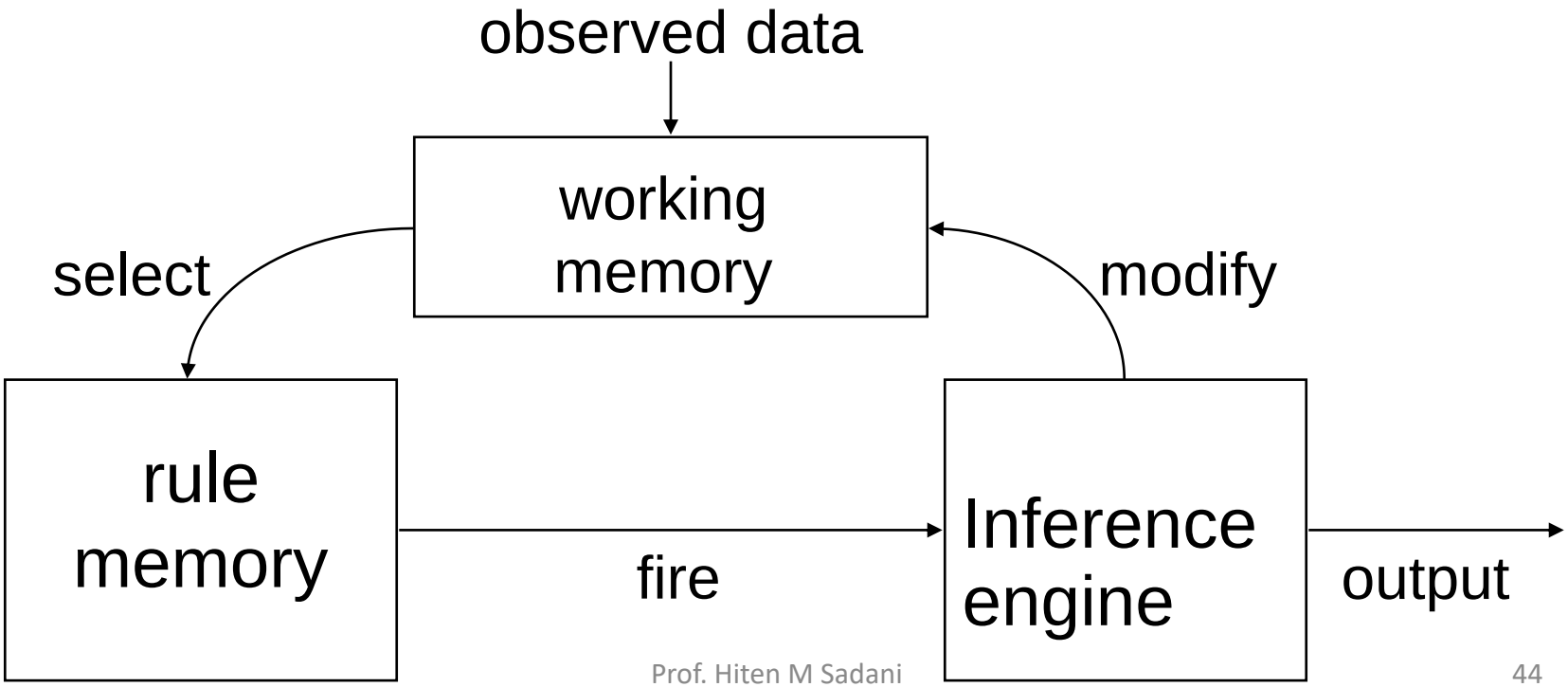
Reasoning with production rules

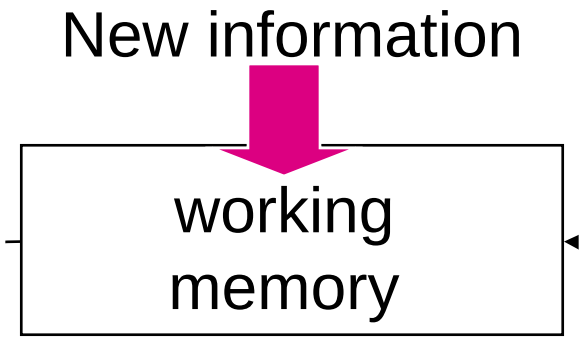
- For example (assuming words beginning with capital letters are variables, and other words are constants):

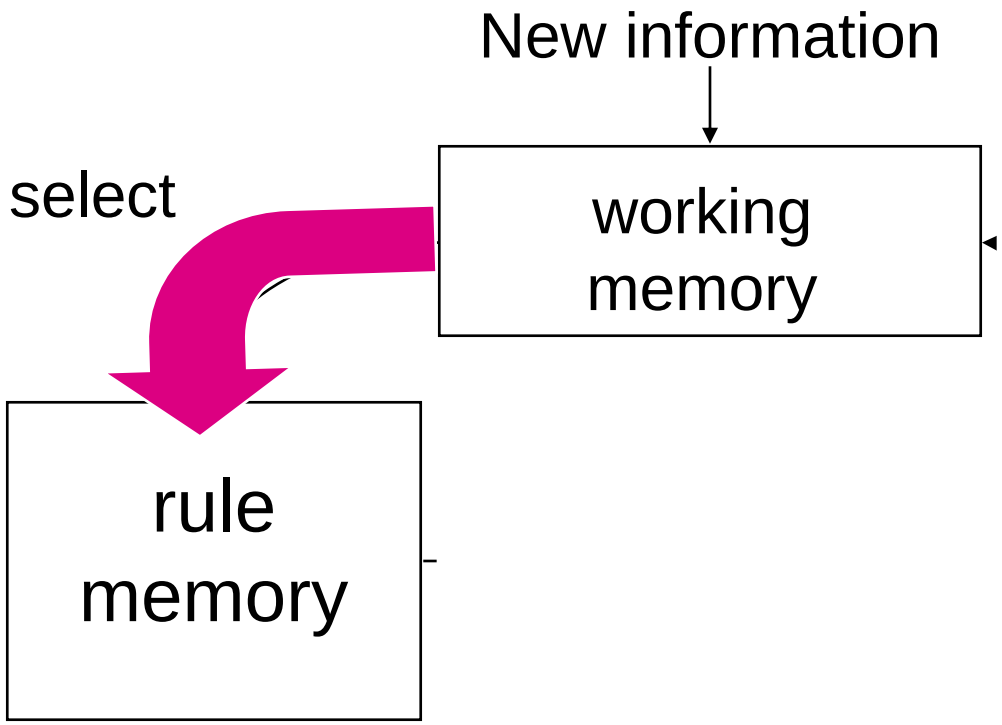
```
if      [Person, age, Number] &
        [Person, employment, none] &
        [Number, greater_than, 18] &
        [Number, less_than, 65]
then    [Person, can_claim,
                unemployment_benefit].
```

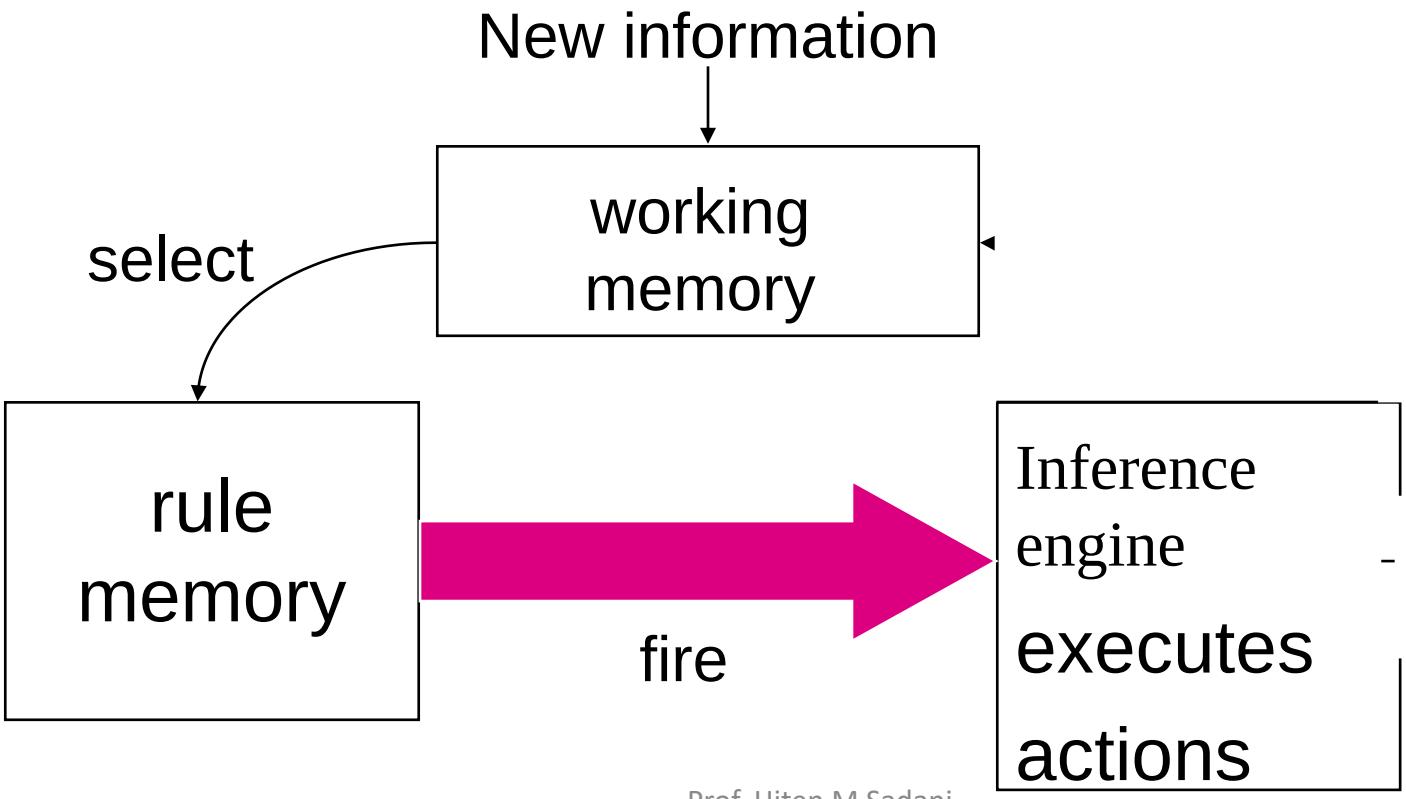
Reasoning with production rules

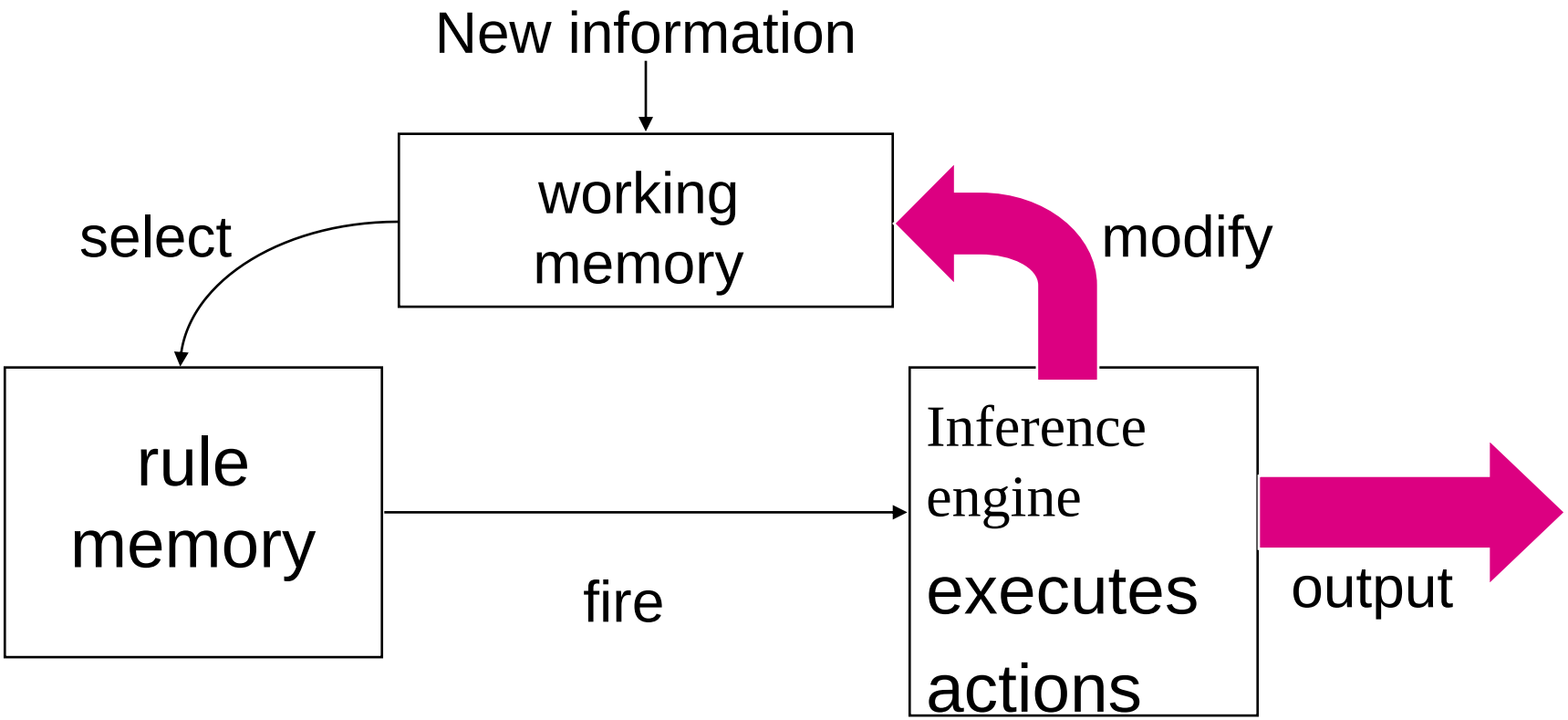
- Architecture of a typical production system:

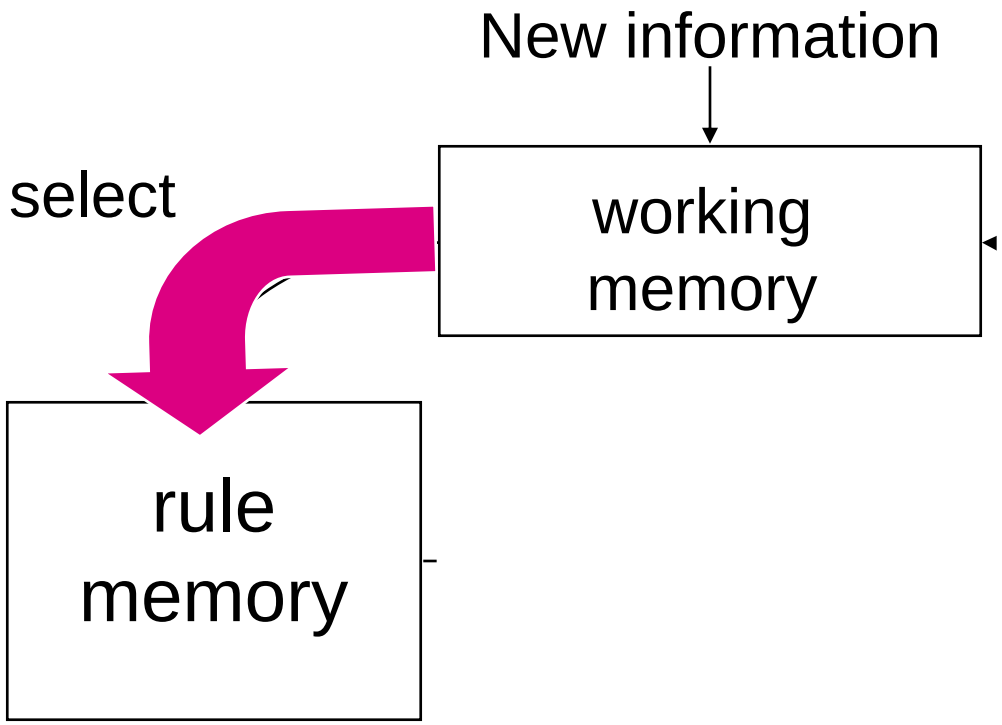


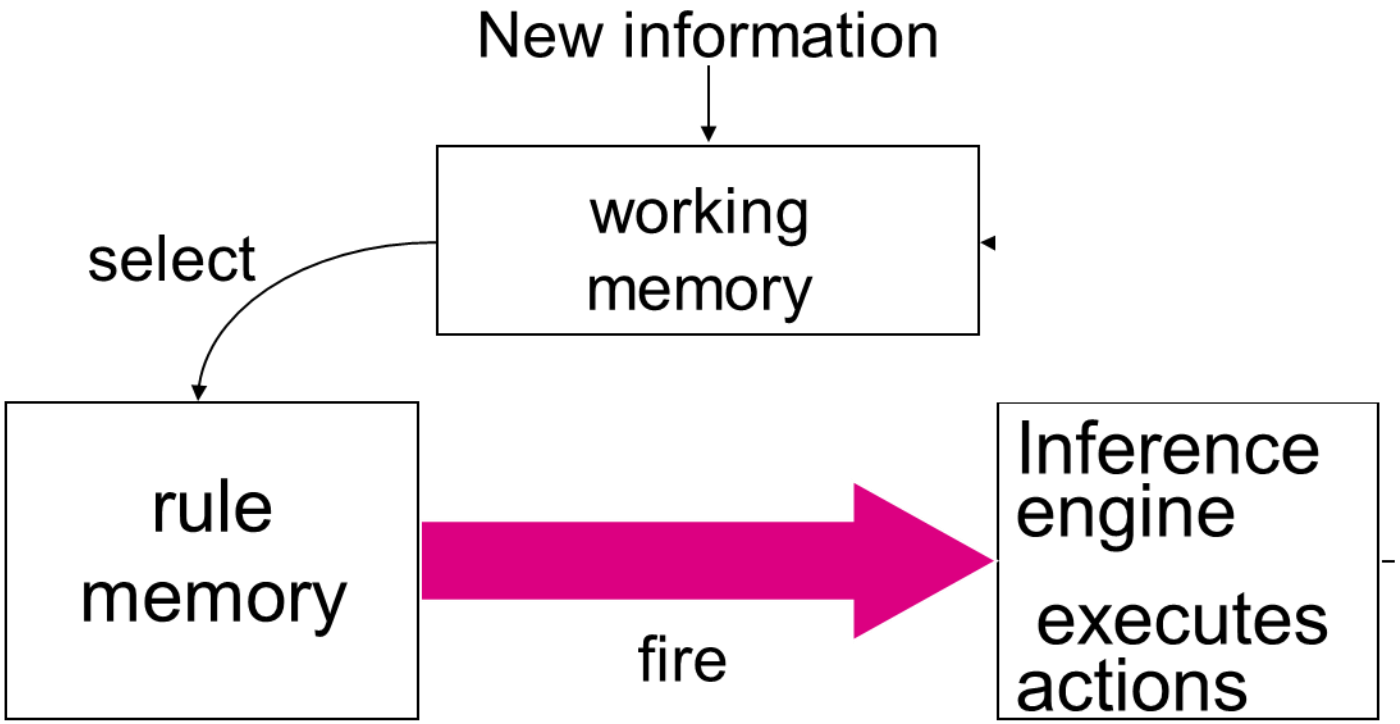

New information
working
memory

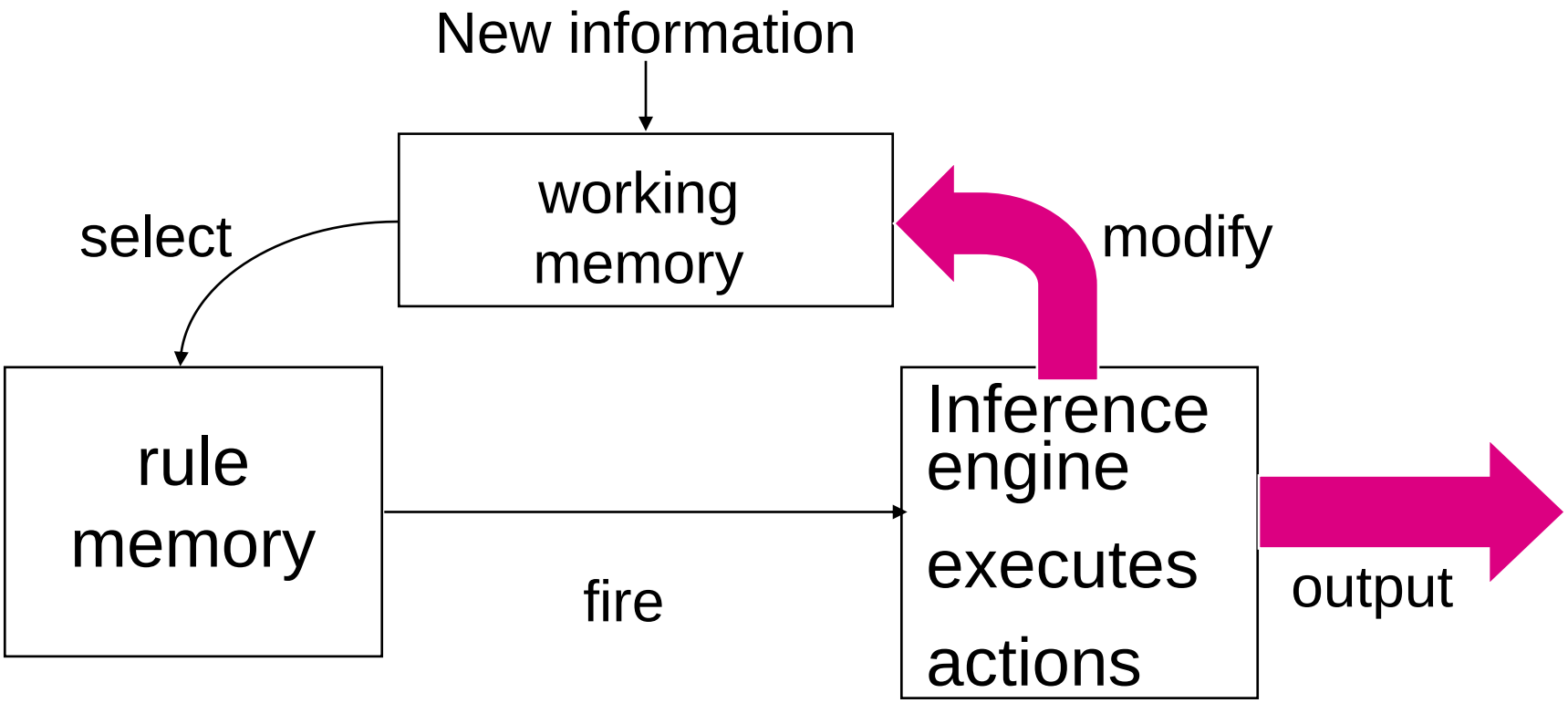

New information
select
working
memory
rule
memory


New information
working memory
select
rule memory
fire
Inference engine
executes actions


New information
working memory
select
modify
rule memory
fire
Inference engine
executes actions
output


New information
select
working
memory
rule
memory


New information
working memory
select
rule memory
fire
Inference engine
executes actions


New information
working memory
select
modify
rule memory
fire
Inference engine
executes
actions
output

Resolution Principle

- Given two clauses C_1 and C_2 with no variables in common, if there is a literal l_1, in C_1 and which is a complement of a literal l_2 in C_2 , both l_1 and l_2 are deleted and a disjuncted C is formed the remaining reduced clauses. The new clauses C is called the resolve of C_1 and C_2.
- Resolution is the process of generating these resolvents from the set of clauses.

Example: (~ PVQ) and (~Q V R)

- We can write

(~ PVQ) , (~Q V R)

~P V R

Refutation

- Resolution produces proofs by refutation
- In other words, to prove a statement, resolution attempts to show that the negation of the statement produces a contradiction with the known statements.

Example of Resolution

- Consider the following clauses:-

  A : P V Q V R

  B: ~ P V Q V R

  C: ~Q V R

- Solution

A: P V Q V R	(Given in the problem)
B : ~P V Q V R	(Given in the problem)
D : Q V R	(Resolvent of A and B)
C: ~Q V R	(Given in the problem)
E: R	(Resolvent of C and D)

Steps:

- Negate the statement to be proved
- Convert given facts into FOL
- Convert FOL into CNF
- Draw Resolution graph/tree

Proof with resolution rule:

- Set of expression F
- Prove: P
- **Procedure:**
  1. Convert F into clauses.
  2. Take ¬P, convert ¬P into clauses. Add to result of Step 1.
  3. Apply resolution rule to produce empty set, i.e., find a contradiction.

Unification

- Any substitution that makes two or more expression equal is called a unifier for the expression.
- Two formulas unify if they can be made identical
- A unification is a function that assigns bindings to variables
- A binding is either a constant, a functional expression or another variable.

Example

- $P(x, x)\ P(A, A) \quad \{x/A\}$
- $P(x, x)\ P(A, B) \quad fail$
- $P(x, y)\ P(A, B) \quad \{x/A\ , y/B\}$
- $P(x, y)\ P(A, A) \quad fail$
- $P(x, y)\ P(A, z) \quad \{x/A\ , y/z\}$

Unification Algorithm

Unification algorithm

Algorithm: Unify(L1, L2)

I. If *L1* or *L2* are both variables or constants, then:

(a) If *L1* and *L2* are identical, then return NIL.

(b) Else if *L1* is a variable, then if *L1* occurs in *L2* then return {FAIL}, else return *(L2/L1)*.

(c) Else if *L2* is a variable, then if *L2* occurs in *L1* then return {FAIL} , else return *(L1/L2)*.

(d) Else return {FAIL}.

2. If the initial predicate symbols in *L1* and *L2* are not identical, then return {FAIL}.

3. If *LI* and *L2* have a different number of arguments, then return {FAIL}.

4. Set *SUBST* to NIL. (At the end of this procedure, *SUBST* will contain all the substitutions used to unify *L1* and *L2*.)

5. For $i \leftarrow 1$ to number of arguments in *L1 :*

(a) Call Unify with the *i*th argument of *L1 and the i*th argument of *L2,* putting result in *S.*

(b) If S contains FAIL then return {FAIL}.

(c) If S is not equal to NIL then:

(i) Apply S to the remainder of both *L1* and *L2.*

(ii) *SUBST:* = APPEND(S, *SUBST).*

6. Return *SUBST.*

Solve: Example

- (Unification) For each pair of atomic sentences, give the most general unifier if it exists, otherwise say "fail":

a. R(A, x), R(y, z)

b. P(A, B, B), P(x, y, z)

c. Q(y, G(A, B)), Q(G(x,x), y)

d. Older(Father(y), y), Older(Father(x), John)

e. Knows(Father(y),y), Knows(x,x)

Solution

- (Unification) For each pair of atomic sentences, give the most general unifier if it exists, otherwise say "fail":

a. R(A, x), R(y, z) y/A, x/z

b. P(A, B, B), P(x, y, z) x/A, y/B, z/B

c. Q(y, G(A, B)), Q(G(x,x), y) fail

d. Older(Father(y), y), Older(Father(x), John) x/y, y/John

e. Knows(Father(y),y), Knows(x,x) fail

FOL into CNF

- Eliminate -> (implies) & <-> (double implies)
- $a \rightarrow b \; so \; \neg a \vee b$
- $a \Leftrightarrow b \; so \; a \rightarrow b \wedge b \rightarrow a$
- Move $\neg$ inwards
  - $\neg(\forall x P) = \exists x \neg P$
  - $\neg(\exists x P) = \forall x \neg P$
  - $\neg(a \vee b) = \neg a \wedge \neg b$
  - $\neg(a \wedge b) = \neg a \vee \neg b$
  - $\neg\neg a = a$
- Rename Variable
- Replace Existential quantifier by skolem constant
  - $\exists x \; Rich(x) = Rich(G1)$
- Drop Universal Quantifier

Example: Problem Statement

1. Ravi likes all kind of food
2. Apples and Chicken are food
3. Anything anyone eats and is not killed is food
4. Ajay eats peanuts and still alive
5. Rita eats everything that Ajay eats

- Prove by resolution that Ravi likes Peanuts using Resolution

Converting given statement into Predicate Logic

1. $\forall x : food(x) \rightarrow likes(Ravi, x)$

2. $food(apple) \wedge food(chicken)$

3. $\forall a \forall b : eats(a,b) \wedge \neg killed(a) \rightarrow food(b)$

4. $eats(Ajay, Peanuts) \wedge alive(Ajay)$

5. $\forall e : \neg killed(e) \rightarrow alive(e)$

6. $\forall d : alive(d) \rightarrow \neg killed(d)$

7. $\forall c : eats(Ajay, c) \rightarrow eats(Rita, c)$

$Conclusion : likes(Ravi, Peanuts)$

Convert into CNF

$1.\neg food(x) \vee likes(Ravi, x)$

$2.\ food(apple)$

$3.\ food(chicken)$

$4.\neg eats(a,b) \vee killed(a) \vee\ food(b)$

$5.eats(Ajay, Peanuts)$

$6.alive(Ajay)$

$7.killed(e) \vee alive(e)$

$8.\neg alive(d) \vee \neg killed(d)$

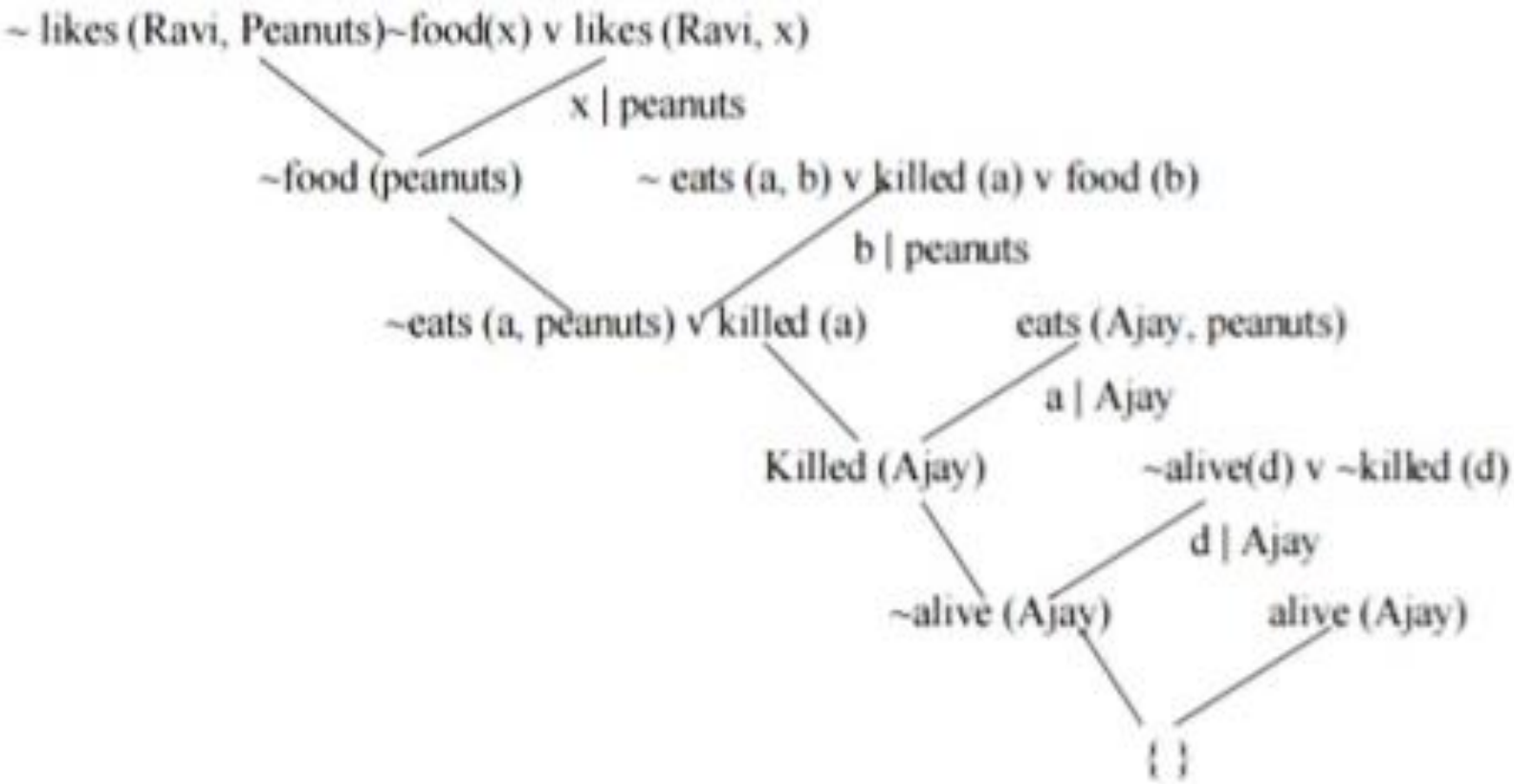
$9.\neg eats(Ajay, c) \vee eats(Rita, c)$

$Conclusion : likes(Ravi, Peanuts)$

Negate Conclusion

$$\neg likes(Ravi, Peanuts)$$

Resolution Tree



Rule of inference	Tautology	Name
$p \to q$ $\underline{p}$ $\therefore q$	$[p \wedge (p \to q)] \to q$	Modus ponens
$\neg q$ $\underline{p \to q}$ $\therefore \neg p$	$[\neg q \wedge (p \to q)] \to \neg p$	Modus tollen
$p \to q$ $\underline{q \to r}$ $\therefore p \to r$	$[(p \to q) \wedge (q \to r)] \to (p \to r)$	Hypothetical syllogism
$p \vee q$ $\underline{\neg p}$ $\therefore q$	$((p \vee q) \wedge \neg p) \to q$	Disjunctive syllogism
$\underline{p}$ $\therefore p \vee q$	$p \to (p \vee q)$	Addition
$\underline{p \wedge q}$ $\therefore p$	$(p \wedge q) \to p$	Simplification
p $\underline{q}$ $\therefore p \wedge q$	$((p) \wedge (q)) \to (p \wedge q)$	Conjunction
$p \vee q$ $\underline{\neg p \vee r}$ $\therefore q \vee r$	$[(p \vee q) \wedge (\neg p \vee r)] \to (p \vee r)$	Resolution

1. It is not sunny this afternoon and it is colder than yesterday.
2. If we go swimming it is sunny.
3. If we do not go swimming then we will take a canoe trip.
4. If we take a canoe trip then we will be home by sunset.
5. We will be home by sunset

p It is sunny this afternoon

q It is colder than yesterday

r We go swimming

s We will take a canoe trip

t We will be home by sunset (the conclusion)

propositions

1. $\neg p \wedge q$
2. $r \rightarrow p$
3. $\neg r \rightarrow s$
4. $s \rightarrow t$
5. t

hypotheses

Using the rules of inference to build arguments

An example

p It is sunny this afternoon

q It is colder than yesterday

r We go swimming

s We will take a canoe trip

t We will be home by sunset (the conclusion)

1. $\neg p \wedge q$
2. $r \rightarrow p$
3. $\neg r \rightarrow s$
4. $s \rightarrow t$
5. t

	Step	Reason
1.	$\neg p \wedge q$	Hypothesis
2.	$\neg p$	Simplification using (1)
3.	$r \rightarrow p$	Hypothesis
4.	$\neg r$	Modus tollens using (2) and (3)
5.	$\neg r \rightarrow s$	Hypothesis
6.	s	Modus ponens using (4) and (5)
7.	$s \rightarrow t$	Hypothesis
8.	t	Modus ponens using (6) and (7)

Rule of inference	Tautology	Name
$p \rightarrow q$ $\underline{p}$ $\therefore q$	$[p \wedge (p \rightarrow q)] \rightarrow q$	Modus ponens
$\neg q$ $\underline{p \rightarrow q}$ $\therefore \neg p$	$[\neg q \wedge (p \rightarrow q)] \rightarrow \neg p$	Modus tollen
$p \rightarrow q$ $\underline{q \rightarrow r}$ $\therefore p \rightarrow r$	$[(p \rightarrow q) \wedge (q \rightarrow r)] \rightarrow (p \rightarrow r)$	Hypothetical syllogism
$p \vee q$ $\underline{\neg p}$ $\therefore q$	$((p \vee q) \wedge \neg p) \rightarrow q$	Disjunctive syllogism
$\underline{p}$ $\therefore p \vee q$	$p \rightarrow (p \vee q)$	Addition
$\underline{p \wedge q}$ $\therefore p$	$(p \wedge q) \rightarrow p$	Simplification
p $\underline{q}$ $\therefore p \wedge q$	$((p) \wedge (q)) \rightarrow (p \wedge q)$	Conjunction
$p \vee q$ $\underline{\neg p \vee r}$ $\therefore q \vee r$	$[(p \vee q) \wedge (\neg p \vee r)] \rightarrow (p \vee r)$	Resolution

Non Production System Architecture

- A Non Production system architecture of certain expert system does not have rule representation scheme.
- It is more structured representation like sematic networks, frames and rule structures decision trees, or even specialized networks like neural networks.

Non Production System Architecture

- They are classified into six categories –
- Associative or semantics architecture
- Frame and rule structures
- Decision network
- Black board system architecture
- Neural networks
- Analogical reasoning

Associative or semantics architecture

- Associative network representations are especially useful in hierarchical representation of knowledge structures for example Binary tree.
- Associative network representations are not a popular form of representation for standard expert systems.
- They are used in natural languages or computer version systems.

Associative or semantics architecture

- One expert system based on the use of associative network representation is CASNET (Casual Associative Network), which was developed at Rutgers for glaucoma one of the leading causes of blindness. Main components of this network are –

1. patient observation
2. Disease categories

Frames and rule structures

- Frame is network of nodes and relations.
- Expert system has been constructed with frame architecture and numbers of building tools that create and manipulate frame structured system have been developed.
- Example- A frame based system is PIP (present illness program) system developed at 1970's.

Frames and rule structures

- This medical knowledge in PIP is organized in frame structures where each frames is composed of categories of slot with names such as typical findings.
- Disease, solution, research, doctors, patients etc.

Decision Architecture

- Decision architecture is similar to decision tree in decision tree we can find initial state(Starting node) and goal state(Last node).

Black Board Architecture

- Black board architecture is special types of knowledge base system. Its differ from forward and backward chaining in production systems the point may be chosen in any way of the given path and searching the tree.
- Black board system have been gaining popularity recently. They have been applied to a number of application areas.
- One of the first application was HEARSAY family of projects, which are speech – understanding systems.

Neural Network Architecture

- Neural networks are large networks. Neural network is used to solve large problem into short form. It is used to solve mathematically and statistical problem.

Analogical Reasoning

- In analogical reasoning is used to match current problem, and if matched then solution can be generated with the previous experience. With some modification such system uses past experience based on promise that human beings use analogical reasoning experimental reasoning to learn and solve complex problems.

Knowledge acquisition and Validation

- **Knowledge acquisition** is the process of adding new knowledge to a knowledge base and refining or otherwise improving knowledge that was previously acquired.
- Acquisition is usually associated with some purpose such as expanding the capabilities of a system or improving its performance at some specified task.
- It is goal oriented creation and refinement of knowledge . It may consist of facts, rules, concepts, procedures, heuristics, formulas, relationships, statistics or other useful information.
- **Validation** is measuring the performance of the expert system.

System Building Tools

- CLIPS
- CLIPS is C Language
- Integrated Production System an expert system shell.
- CLIPS uses a LISP

(List Processing language)
like notation to enter rules.

```
CLIPS>  (defrule rule1
(elevator ?floor_now)
(button ?floor_now)
=>
(assert (open_door)))
CLIPS>  (defrule rule2
(elevator ?floor_now)
(button ?other_floor)
=>
(assert (goto ?other_floor)))
CLIPS>  (assert (elevator floor1))
==> f-0 (elevator floor1)
<Fact-0>
CLIPS> (assert (button floor3))
==> f-1 (button floor3)
<Fact-1>
<CLIPS> (run)
==>f-2  (goto floor3)
```

Limitations of Expert Systems

- Uncertainty = having limited knowledge (more than possible outcomes)
- Both human experts and expert systems must be able to deal with uncertainty.
- Limitation 1: most expert systems deals with shallow knowledge than with deep knowledge.
- Shallow knowledge – based on empirical and heuristic knowledge.
- Deep knowledge – based on basic structure, function, and behavior of objects.

Limitations of Expert Systems

- **Limitation 2**: typical expert systems cannot generalize through analogy to reason about new situations in the way people can.
- **Solution 1 for limitation 2**: repeating the cycle of interviewing the expert.
- **Limitation raised form Solution 1**: A knowledge acquisition bottleneck results from the time consuming and labor intensive task of building an expert system.

Early Expert Systems

- DENDRAL – used in chemical mass spectroscopy to identify chemical constituents
- MYCIN – medical diagnosis of illness
- DIPMETER – geological data analysis for oil
- PROSPECTOR – geological data analysis for minerals
- XCON/R1 – configuring computer systems

Uncertainty in Expert System

- In Expert systems the word uncertainty is related to the working with inexact data, imprecise information, handling identical situation, reliability of the results etc.
- It is not possible to determine whether an assertion in the model is true or false.
- For example, there might be uncertainty about the fact - **The height of plant is 38.**

Uncertainty in Expert System

- There are several sources of uncertainty in rules:
  - Uncertainty related to individual rules
  - Uncertainty due to conflict resolution
  - Uncertainty due to incompatibility of rules

Approximate Reasoning

- This is theory of uncertainty based on fuzzy logic and concerned with quantifying and reasoning using natural language where words have ambiguous meaning.
- Fuzzy logic is a superset of conventional logic – extended to handle partial truth.
- Soft-computing means computing not based on classical two-valued logics – includes fuzzy logic, neural networks, and probabilistic reasoning.