ARTIFICIAL INTELLIGENCE UNIT-3

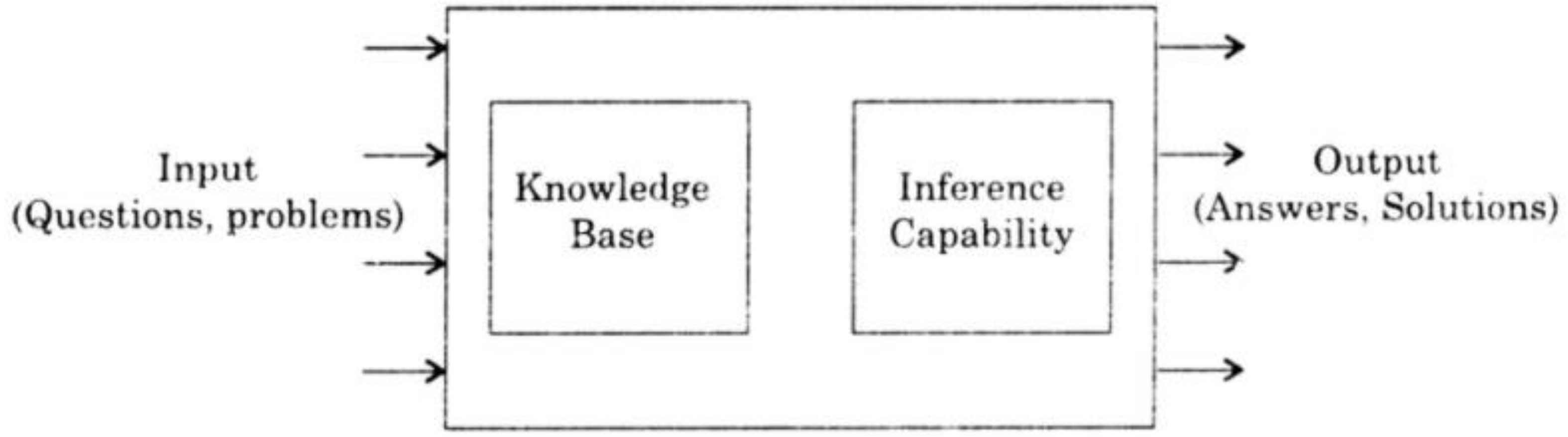
HOW AI TECHNIQUES HELP COMPUTERS TO BE SMARTER?

- Computers can not have experience but it can study and learn.
- For this it should have knowledge.
- What is knowledge?
- Knowledge is more than simply data or information.
- Knowledge consists of
  - Facts
  - Concepts
  - Theories
  - Procedure and relationship between them

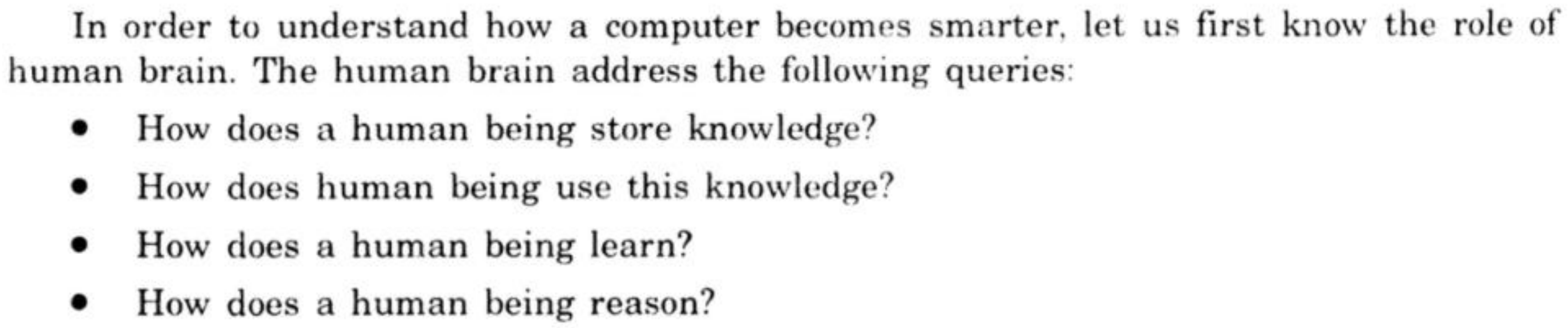
  All these entities form knowledge base

HOW AI TECHNIQUES HELP COMPUTERS TO BE SMARTER?

- When AI techniques(Search Techniques) are applied to this database, a smarter computer results.
- This smarter computer can reason, take decision, make judgement, etc.



HOW AI TECHNIQUES HELP COMPUTERS TO BE SMARTER?



In order to understand how a computer becomes smarter, let us first know the role of human brain. The human brain address the following queries:

- How does a human being store knowledge?
- How does human being use this knowledge?
- How does a human being learn?
- How does a human being reason?

- The art of performing these actions collectively is the aim of AI and is called cognitive science.

AI TECHNIQUES (SEARCH KNOWLEDGE)

- Knowledge can be defined as the body of facts and principles accumulated by humankind or the act, fact or state of knowing.
  - Example, knowledge stored in complex structures of interconnected neurons. The structures correspond to symbolic representations of the knowledge possessed by the organisms, the facts, rules, and so on.
- Types:
  1. Procedural or Operational Knowledge
     - Steps to solve quadratic equation
  2. Declarative or Relational Knowledge
     - Facts about the world
  3. Heuristic Knowledge
     - Tricks, strategy, experience to simplify the solution to problems.

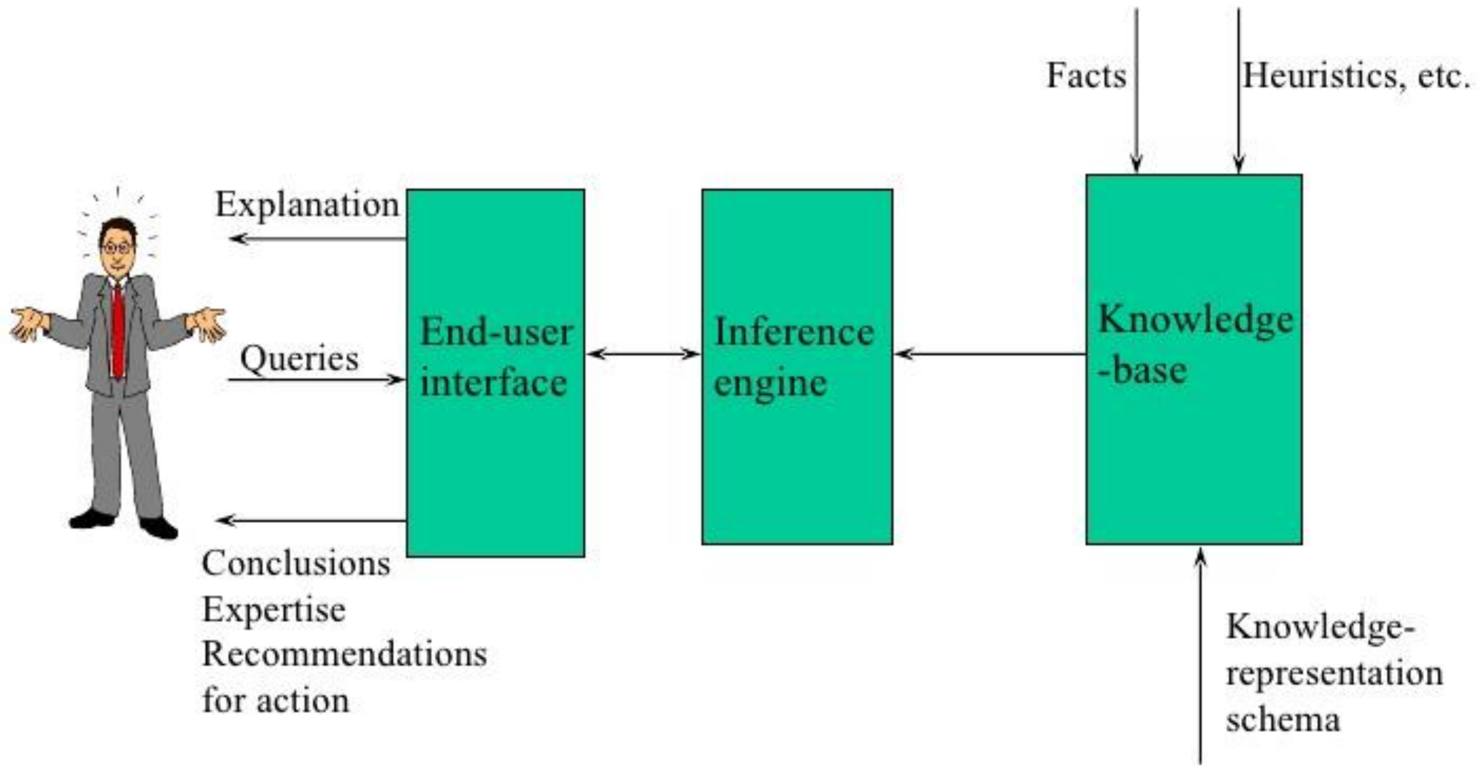
KNOWLEDGE BASED SYSTEM (KBS)

- Those systems that depend on a rich base of knowledge to perform difficult tasks are known as Knowledge based Systems
- Three main Components of KBS
  1. Input-Output Unit
  2. Inference Control Unit
  3. Knowledge Base

KBS COMPONENTS

AI & KBS

3. KBS Architecture



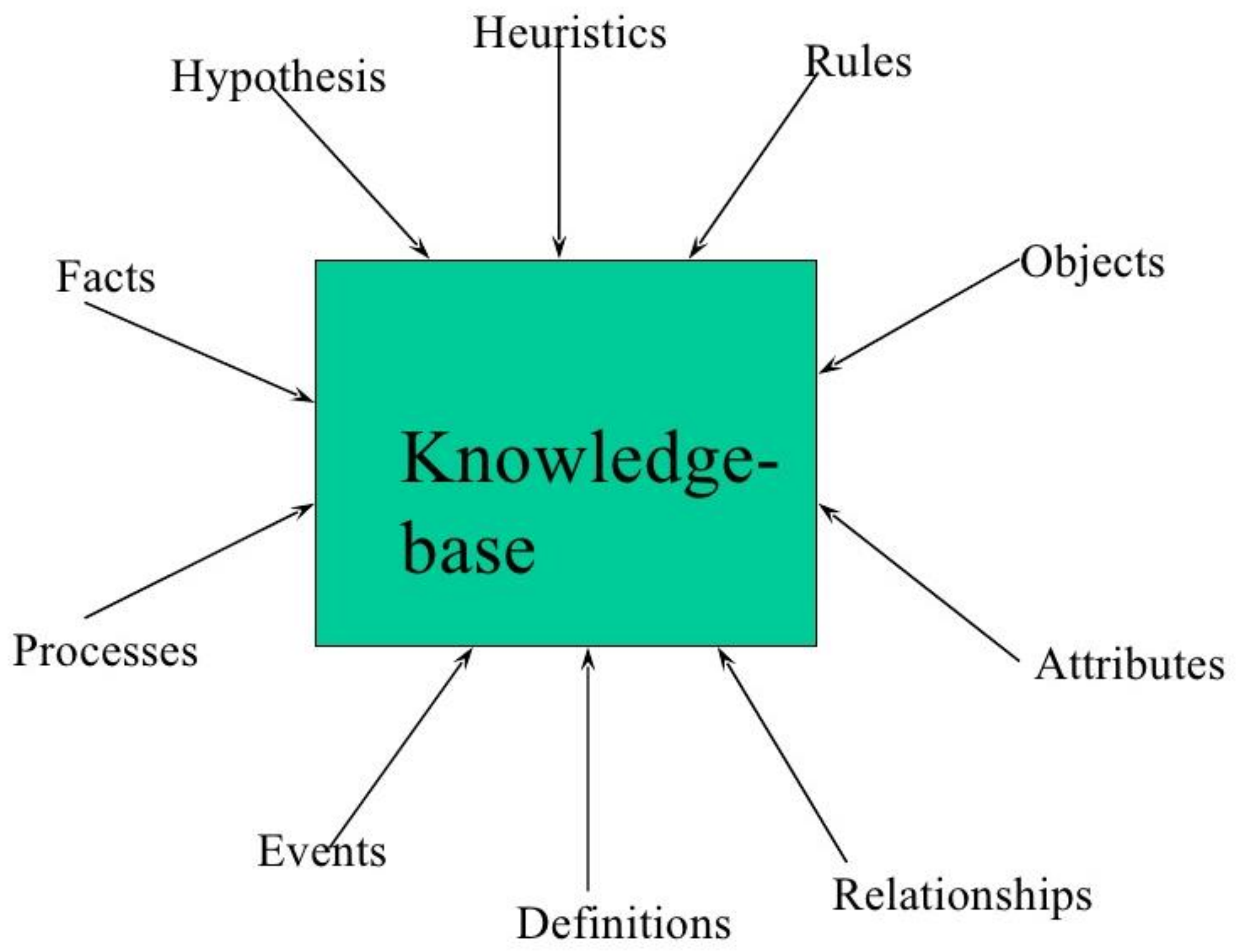
KBS - INFERENCE ENGINE

- It is the component of the system that applies logical rules to the knowledge base to deduce new information
- Two modes:
  1. Forward chaining: starts with the known facts and asserts new facts
  2. Backward chaining: starts with goals and works backward to determine what facts must be asserted so that the goals can be achieved.
- Example of Forward Chaining,
  - B: The road is wet
  - A=>B If it is raining, the road is wet
  - A: It is raining
- Example of Backward Chaining,
  - A: It is raining
  - A=>B If it is raining, the road is wet
  - B: The road is wet

KBS

- Heuristic rather than algorithmic
- Highly specific domain Knowledge
- Knowledge is separated from how it is used
- KBS = knowledge-base + inference engine

(1) Knowledge-base



HOW TO ACQUIRE KNOWLEDGE?

- Knowledge may be acquired from sources like textbooks, references, reports, technical research papers and so on and to be useful, it should be accurate, complete, inconsistent and so on.
- KBS depends on a high quality knowledge for their success.

EXAMPLES OF KBSS

(1) DENDRAL (chemical)

- DENDRAL infers Molecular structure given mass spectral Data
- Expert System
- Like expert Chemist

(2) MYCIN (medicine)

- Assist internists in diagnosis and treatment of infectious diseases
- Given patient data(incomplete & inaccurate) MYCIN gives interim indication of organisms that are most likely causes of infection & drugs to control disease.

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

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.

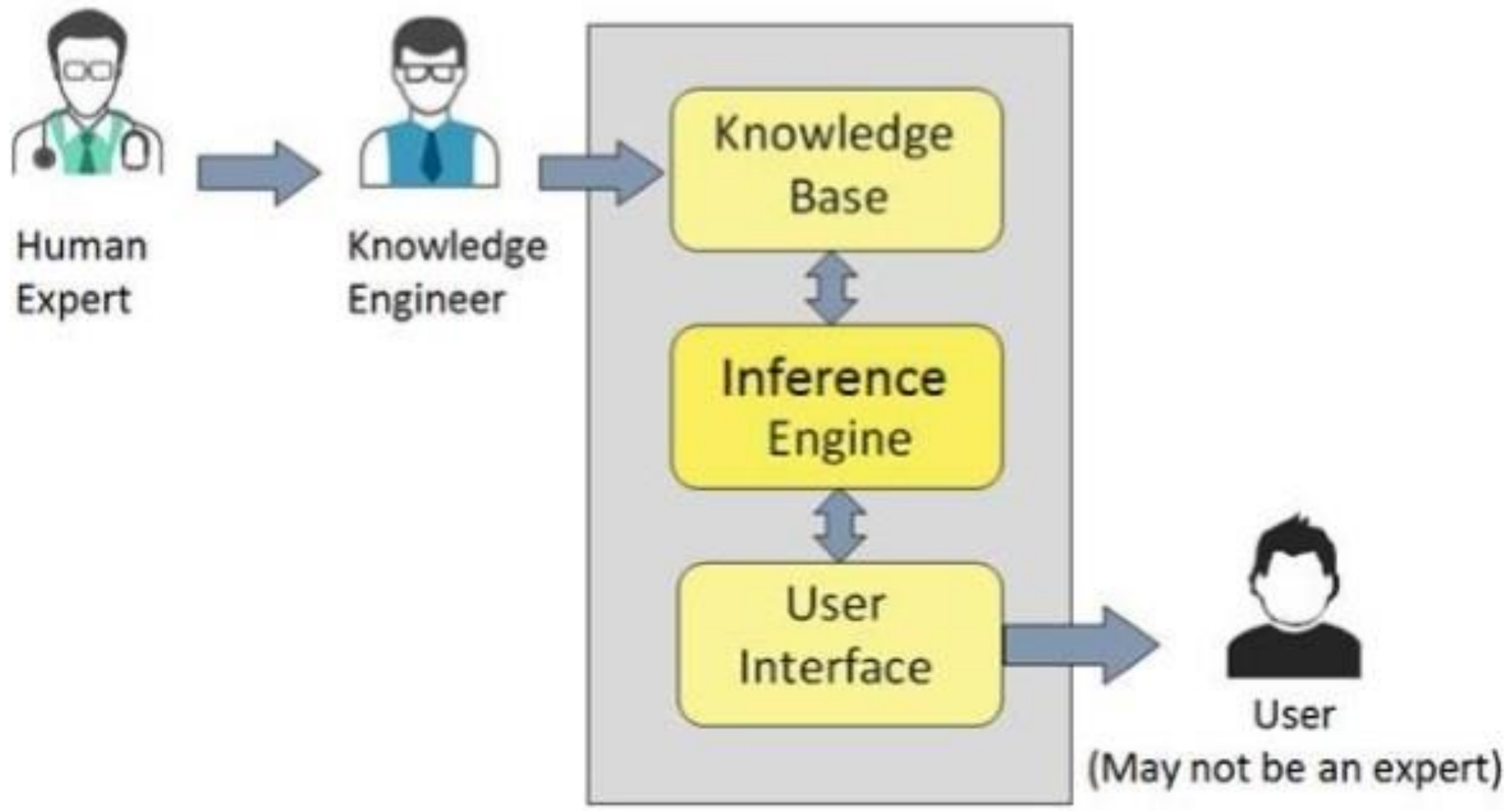
PROMINENT EXPERT SYSTEMS

- 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

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.

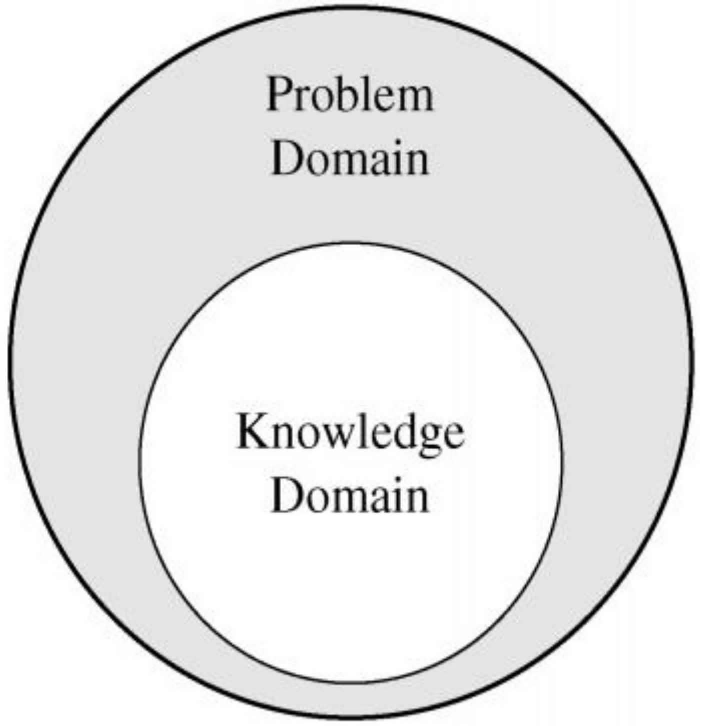
BASIC EXPERT SYSTEM ARCHITECTURE



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 ARCHITECTURE

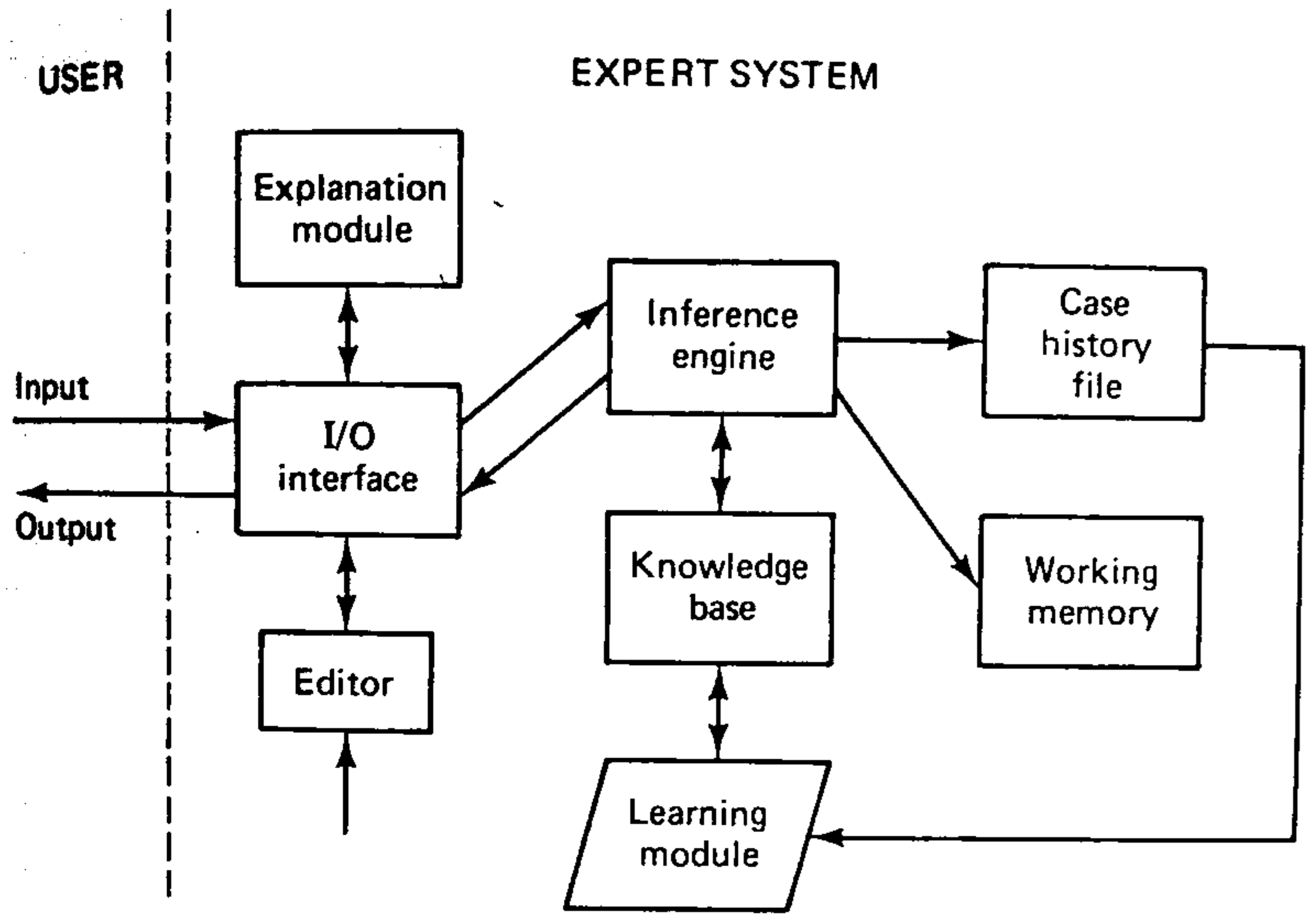
- There are many architectures, on which expert systems are built upon.

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

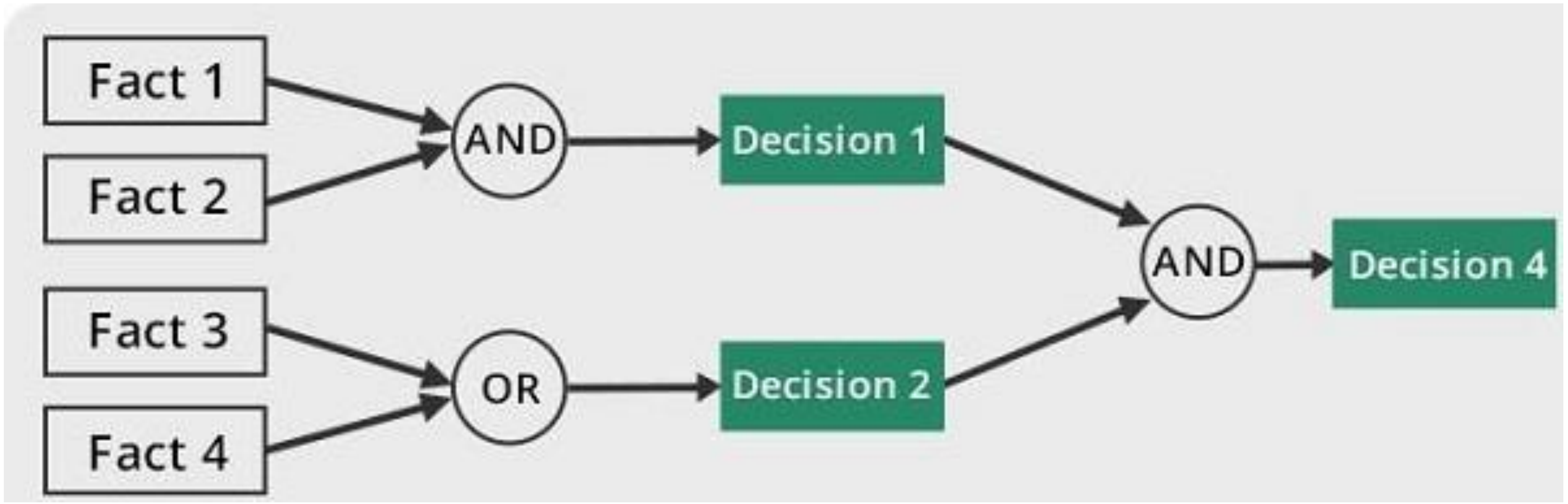
- Inferencing engines for rule-based systems generally work by either forward or backward chaining of rules. Two strategies are:

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.

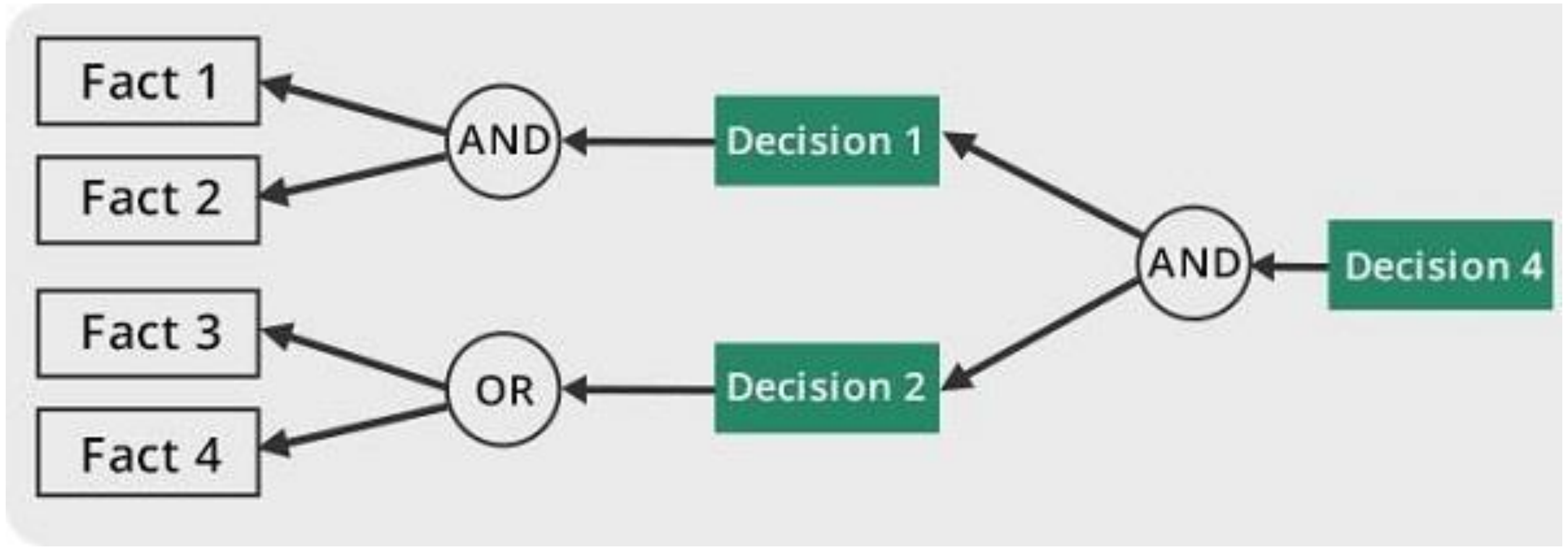
FORWARD CHAINING



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.

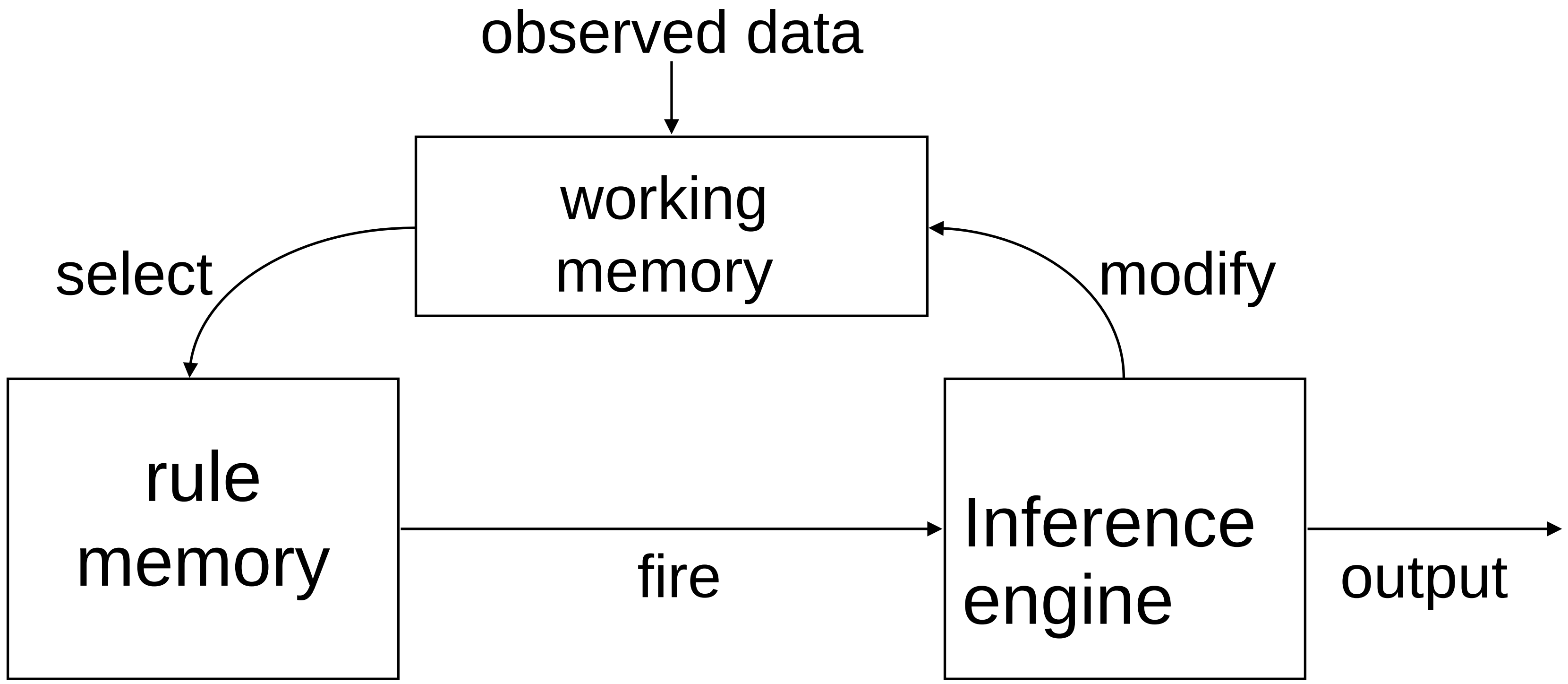
REASONING WITH PRODUCTION RULES

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

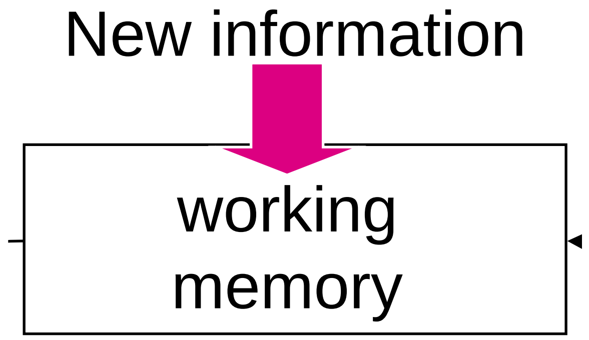
```
if      [Person, age, Number] &
        [Person, employment, none] &
        [Number, greater_than, 18] &
        [Number, less_than, 65]
then    [Person, can_claim,
                unemployment_benefit].
```

REASONING WITH PRODUCTION RULES

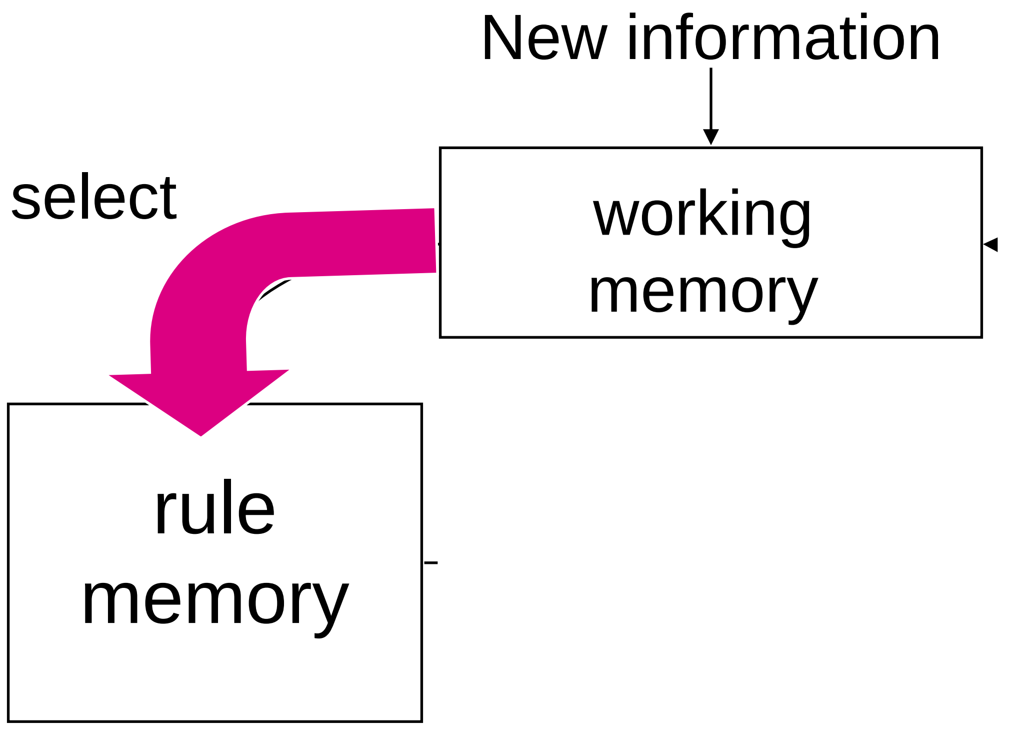
- Architecture of a typical production system:



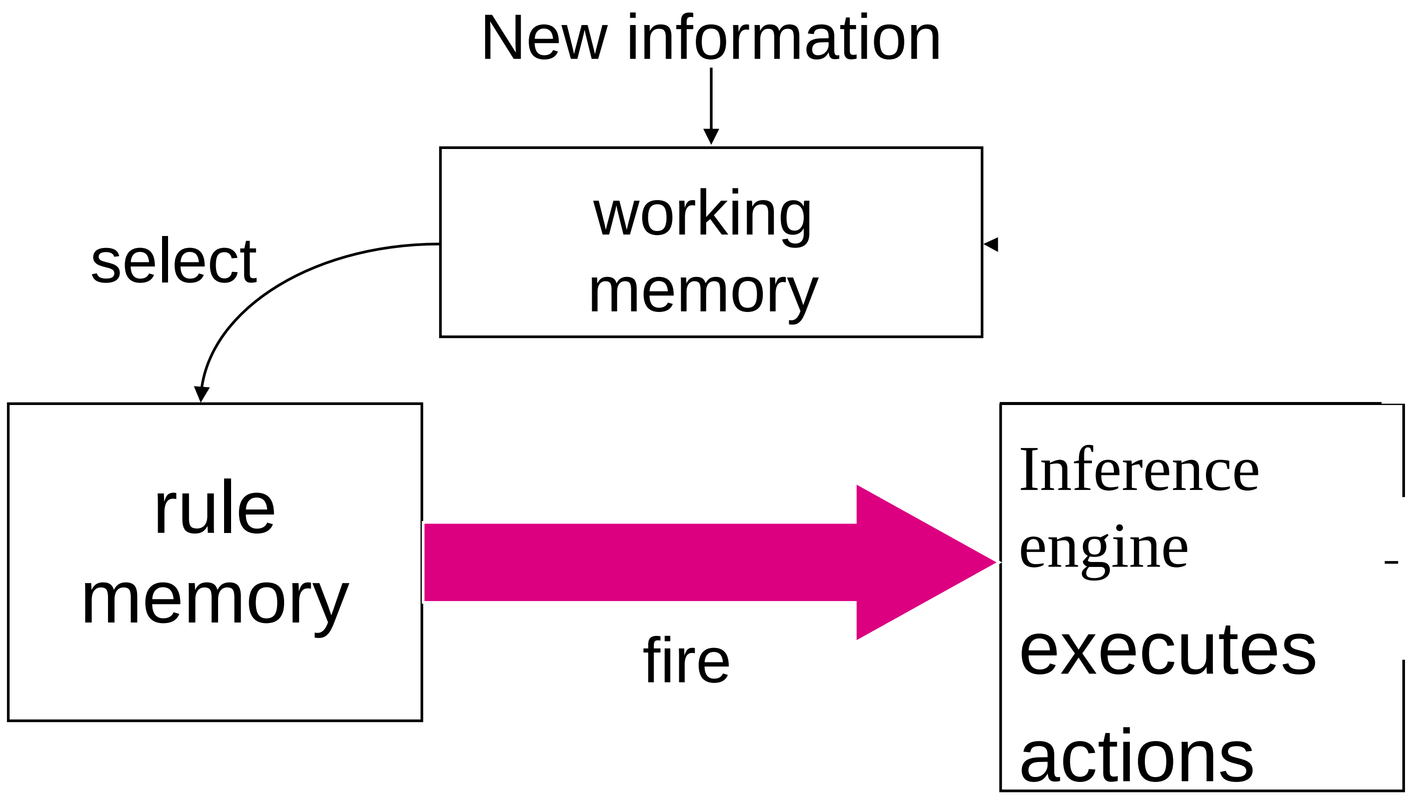
REASONING WITH PRODUCTION RULES



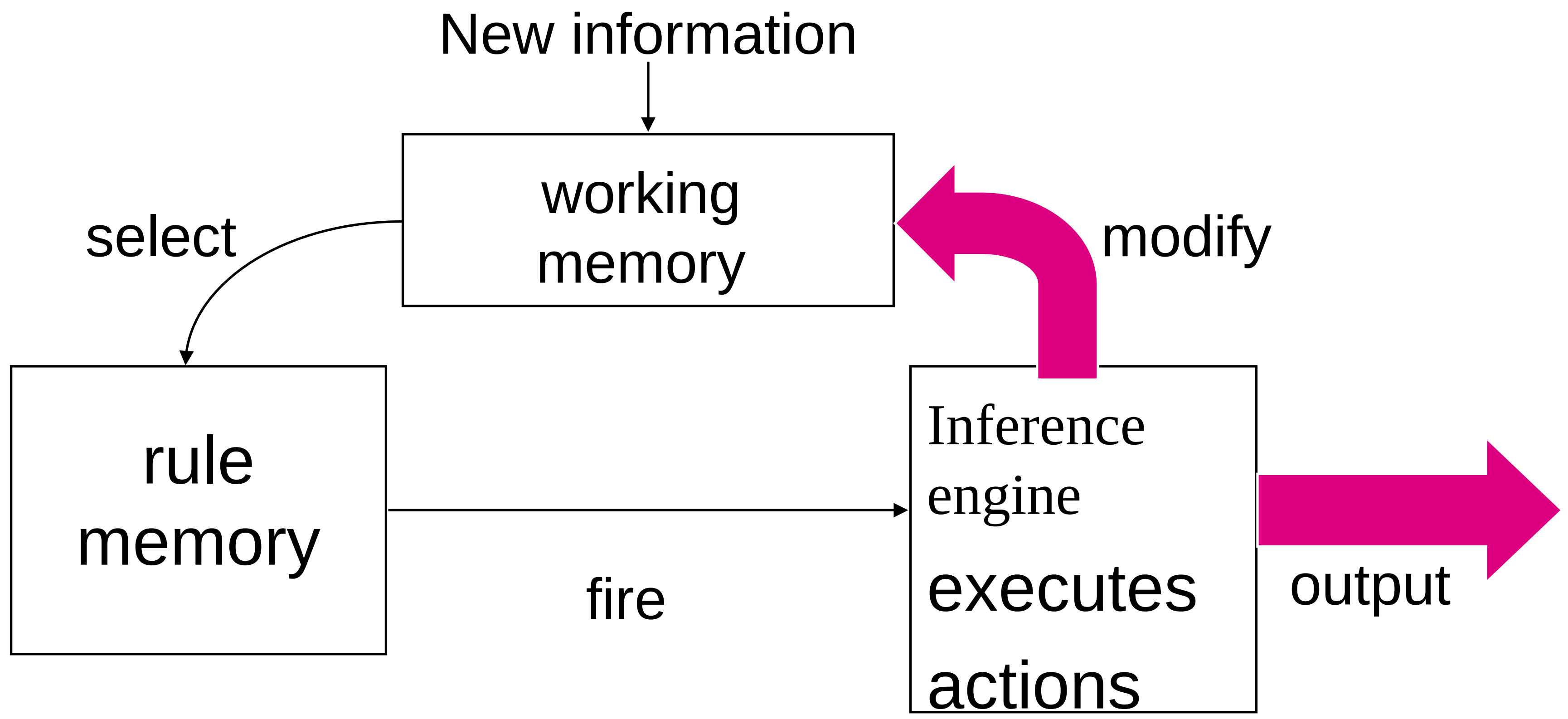
REASONING WITH PRODUCTION RULES



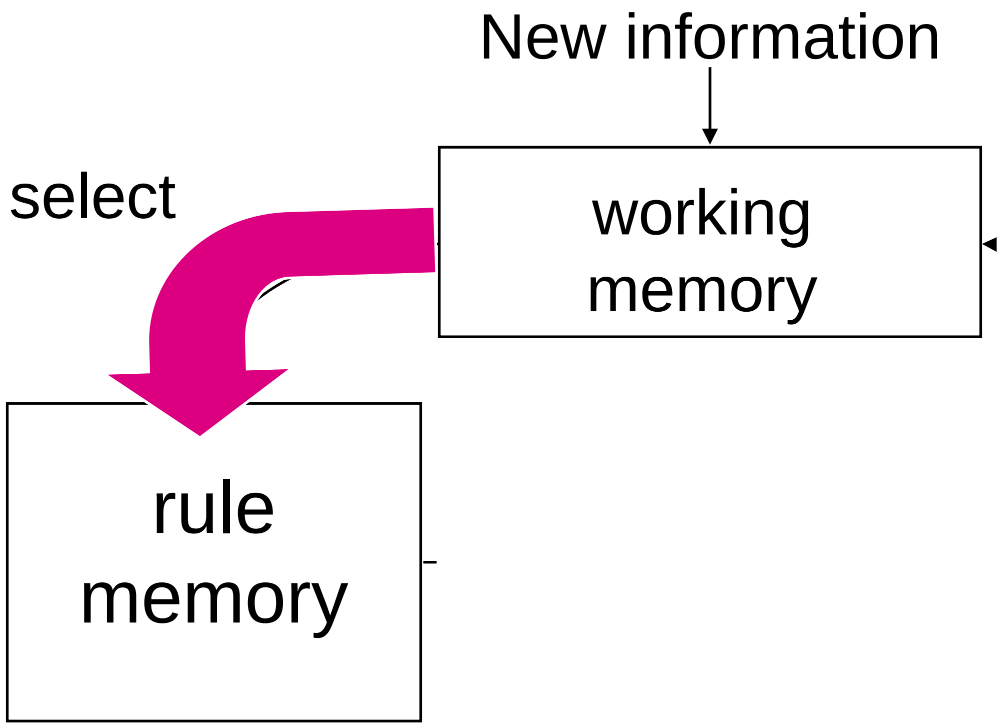
REASONING WITH PRODUCTION RULES



REASONING WITH PRODUCTION RULES

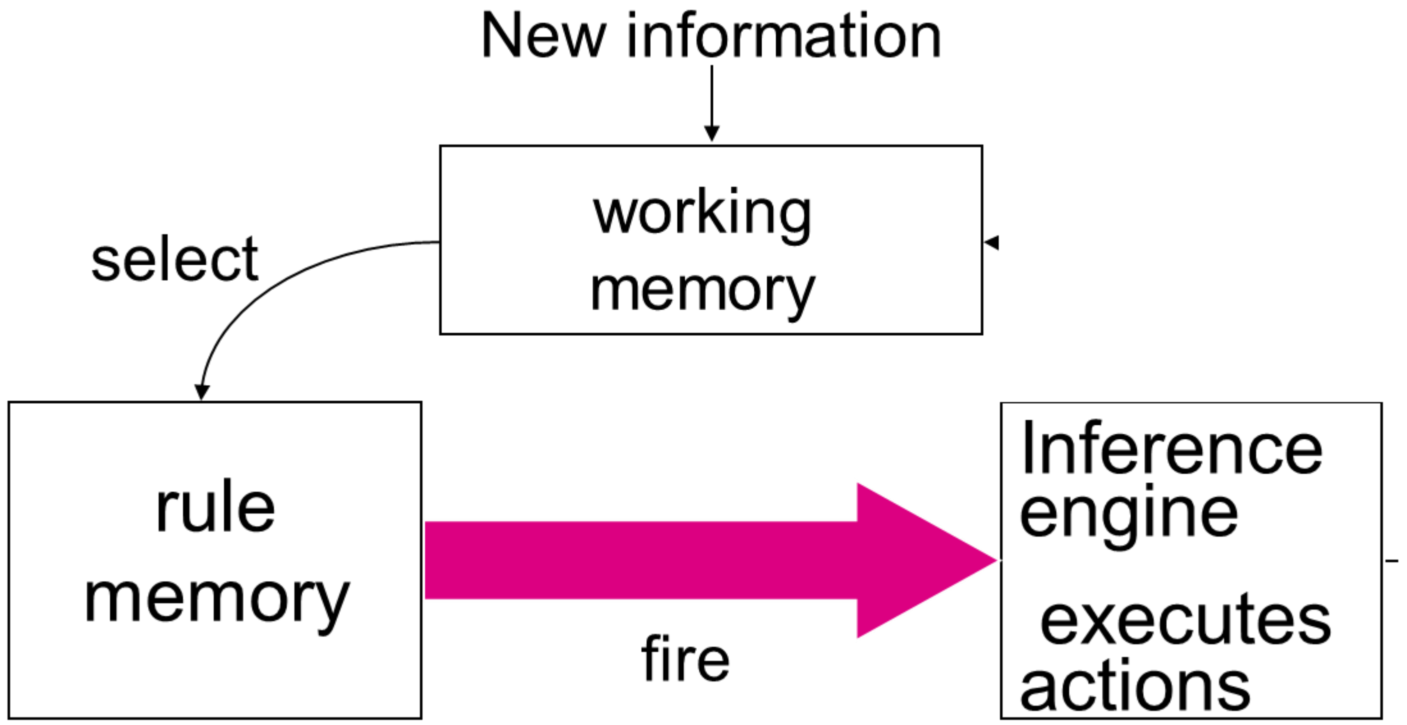


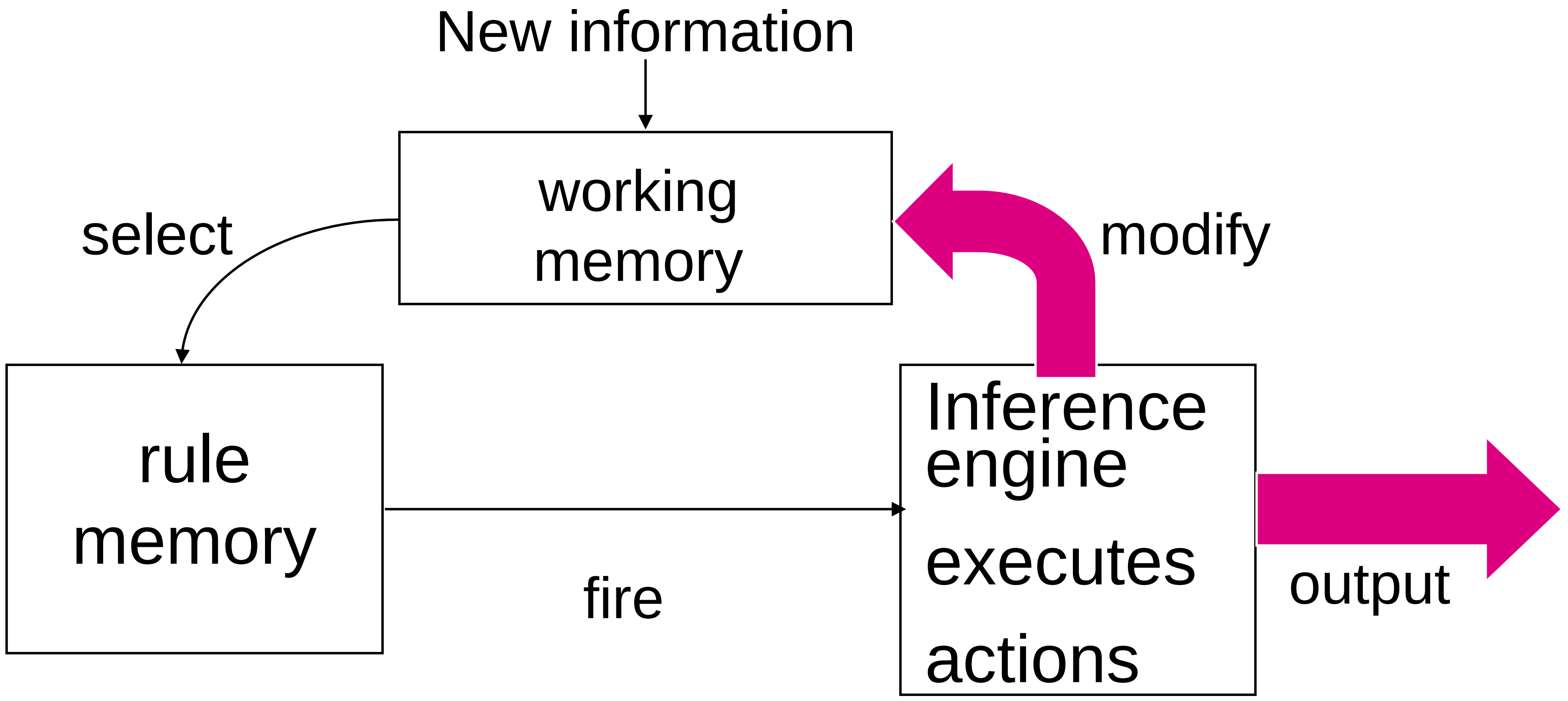
REASONING WITH PRODUCTION RULES



REASONING WITH PRODUCTION RULES

- Architecture of a typical production system




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

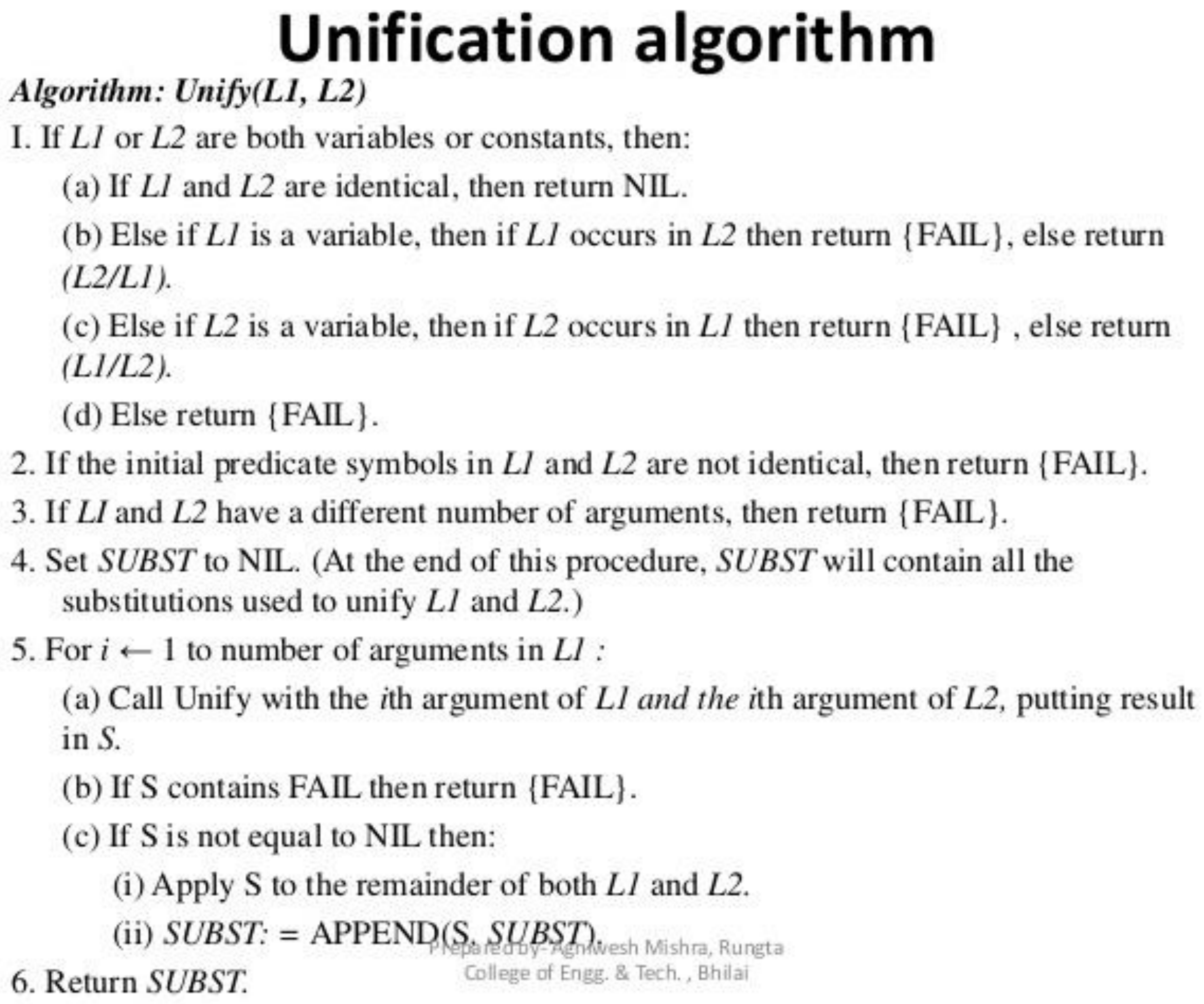
UNIFICATION

- 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 ← 1 to number of arguments in L1 :
    (a) Call Unify with the ith argument of L1 and the ith 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

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

$2. food(apple)$

$3. food(chicken)$

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

$5. eats(Ajay, Peanuts)$

$6. alive(Ajay)$

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

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

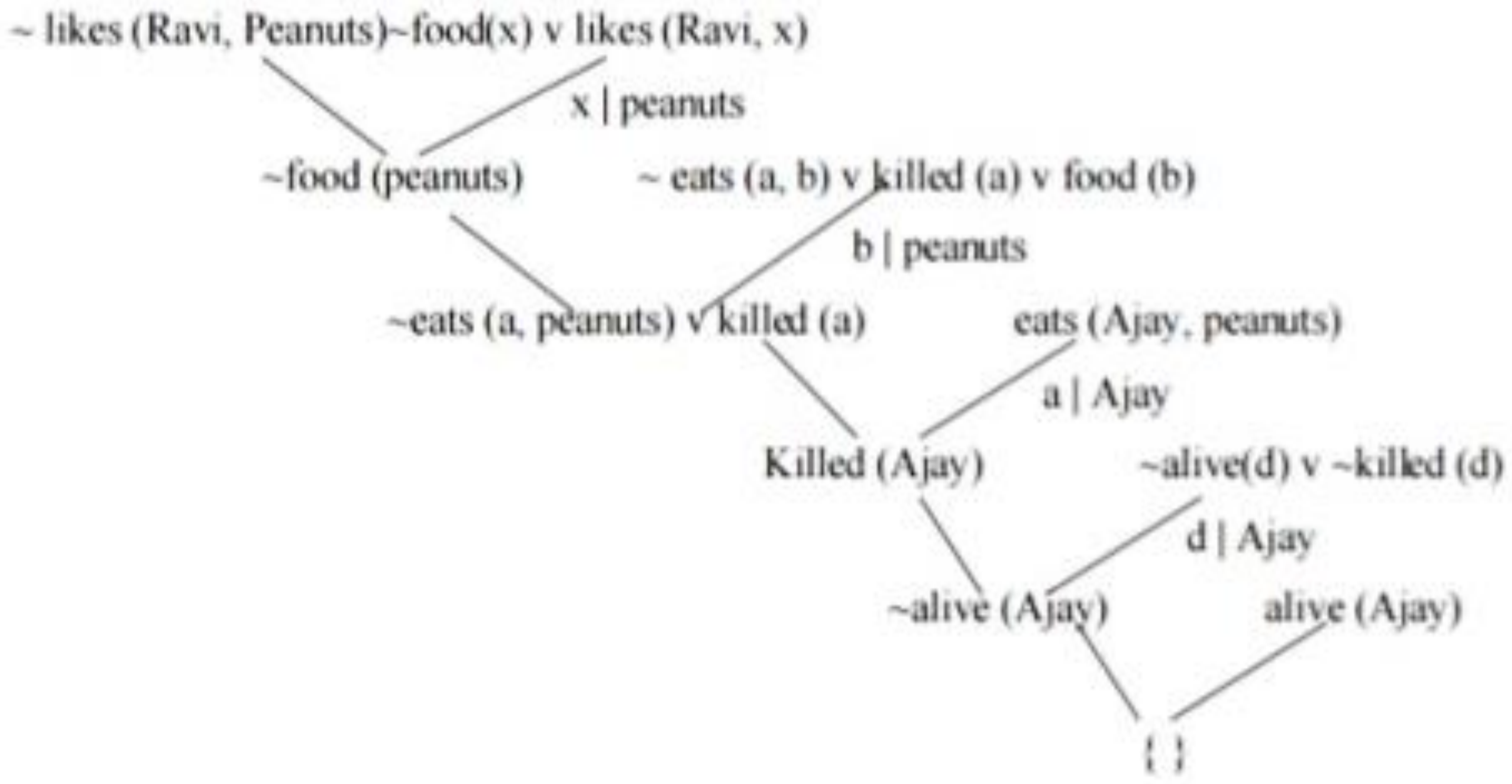
$9. \neg eats(Ajay, c) \vee eats(Rita, c)$

$Conclusion: likes(Ravi, Peanuts)$

NEGATE CONCLUSION

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

RESOLUTION TREE



EXAMPLE

1. Hari likes all kind of food
2. Apples and bananas are food
3. Anything anyone eats and is not killed is food
4. Ajay eats peanuts and still alive
5. Hari eats everything Ram eats

- Prove that Hari likes Peanuts

FOL

$$\forall x: food(x) \rightarrow likes(Hari, x)$$

$$food(apple)$$

$$food(chicken)$$

$$\forall x \forall y: eats(x, y) \wedge \neg killed(x) \rightarrow food(y)$$

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

$$\forall x: \neg killed(x) \rightarrow alive(x)$$

$$\forall x: alive(x) \rightarrow \neg killed(x)$$

$$\forall x: eats(Ram, x) \rightarrow eats(Hari, x)$$

PROVE

- Want to prove

$$likes(Hari, Peanuts)$$

EXAMPLE

1. Mann only likes easy courses.
2. IT courses are hard.
3. All the courses in the CE department are easy.
4. AI is a course in CE.
5. Derive: What course would Mann like?

EXAMPLE: (FOL FORM)

1. forall x easy(x) -> likes(Mann,x)
2. forall x ITCourse(x) -> ~easy(x)
3. forall x CECourse(x) -> easy(x)
4. CECourse(AI)

- The conclusion is encoded as likes(Mann,x).

SOLUTION (FOL TO CNF FORM):

(1) ~easy(x) V likes(mann,x)

(2) ~ITCourse(x) V ~easy(x)

(3) ~CECourse(x) V easy(x)

(4) CECourse(AI)

(5) ~likes(mann,x)

SOLUTION (RESOLUTION):

(1) ~easy(x) V likes(mann,x)

(2) ~ITCourse(x) V ~easy(x)

(3) ~CECourse(x) V easy(x)

(4) CECourse(AI)

(5) ~likes(mann,x)

(6) ~easy(x). **(From 1&5)**

(7) ~CECourse(x). **(From 3&6)**

(8) Empty clause **(From 4&7, x/AI);**

SOLUTION (RESOLUTION):

- the substitution x/ AI is produced by the unification algorithm which says that the only wff (well formed formula) of the form likes(mann,x) which follows from the premises is likes(mann, AI).
- Thus, resolution gives us a way to find additional assumptions (in this case x= AI) which make our theorem true.

- Consider the following statements:
- a) Anyone passing his AI exams and winning the lottery is happy.
- b) Anyone who studies or is lucky can pass his AI exam.
- c) Ravi did not study but he is lucky.
- d) Anyone who is lucky wins the lottery.
- Prove that Ravi is happy! using resolution tree.

- Step 1: First-Order Predicate Logic representation:

1. $\forall x\ (passAIExam(x) \wedge winsLottery(x) \rightarrow happy(x))$
2. $\forall x\ (studies(x) \vee lucky(x) \rightarrow passAIExam(x))$
3. $\neg studies(\text{Ravi}) \wedge lucky(\text{Ravi})$
4. $\forall x\ (lucky(x) \rightarrow winsLottery(x))$

- Step 2: Conversion to Conjunctive Normal Form (CNF) without quantifiers:
- 1. (¬passAIExam(y) ∨ ¬winsLottery(y) ∨ happy(y))
- 2. ¬studies(z) ∨ passAIExam(z)
- 3. ¬lucky(z) ∨ passAIExam(z)
- 4. ¬studies(Ravi)
- 5. lucky(Ravi)
- 6. (¬lucky(w) ∨ winsLottery(w))

The rules of inference

Rule of inference	Tautology	Name
$\begin{array}{l} p \to q \\ \underline{p \qquad} \\ \therefore q \end{array}$	$[p \wedge (p \to q)] \to q$	Modus ponens
$\begin{array}{l} \neg q \\ \underline{p \to q} \\ \therefore \neg p \end{array}$	$[\neg q \wedge (p \to q)] \to \neg p$	Modus tollen
$\begin{array}{l} p \to q \\ \underline{q \to r} \\ \therefore p \to r \end{array}$	$[(p \to q) \wedge (q \to r)] \to (p \to r)$	Hypothetical syllogism
$\begin{array}{l} p \vee q \\ \underline{\neg p \quad} \\ \therefore q \end{array}$	$((p \vee q) \wedge \neg p) \to q$	Disjunctiv e syllogism
$\begin{array}{l} \underline{p \qquad} \\ \therefore p \vee q \end{array}$	$p \to (p \vee q)$	Addition
$\begin{array}{l} \underline{p \wedge q} \\ \therefore p \end{array}$	$(p \wedge q) \to p$	Simplifica tion
$\begin{array}{l} p \\ \underline{q \qquad} \\ \therefore p \wedge q \end{array}$	$((p) \wedge (q)) \to (p \wedge q)$	Conjunctio n
$\begin{array}{l} p \vee q \\ \underline{\neg p \vee r} \\ \therefore q \vee r \end{array}$	$[(p \vee q) \wedge (\neg p \vee r)] \to (p \vee r)$	Resolution

Using the rules of inference to build arguments

An example

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)

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

$\uparrow$ propositions

$\uparrow$ 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.	$\neg p$	Simplifica tion 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
$p \vee q$ $\underline{\neg p}$ $\therefore q$	$((p \vee q) \wedge \neg p) \rightarrow q$	Disjunctiv e syllogism
$\underline{p}$ $\therefore p \vee q$	$p \rightarrow (p \vee q)$	Addition
$\underline{p \wedge q}$ $\therefore p$	$(p \wedge q) \rightarrow p$	Simplifica tion
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

THANKS!