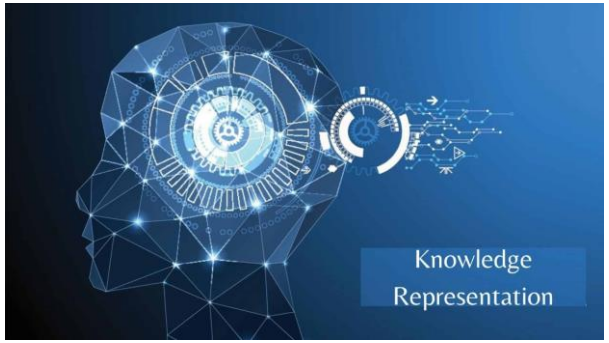
# CHAPTER II

# KNOWLEDGE REPRESENTATION




- Propositional logic
- Predicate logic
- Production rules
- Semantic networks
- Expert systems

# KNOWLEDGE REPRESENTATION in AI

In AI, knowledge refers to the facts, rules, and relationships that an intelligent system uses to understand the world, reason about situations, and make decisions.

**Knowledge has meaning and can be used for inference**

# KNOWLEDGE vs DATA vs INFORMATION

- **Data:** Raw, Unprocessed facts
- **Information:** Organized data
- **Knowledge:** Information combined with rules (usable for reasoning)

| Data | Information | Knowledge |
|---|---|---|
| 38, Karim, cough | Karim has a temperature of 38°C and coughs | **IF** temperature > 37.5 **AND** cough **THEN** possible flu |

# FORMALISMS

knowledge is **formally represented** so that a computer can **reason with it**

- **Logic**
- **Rules**
- **Semantic nets**
- **Frames**
- **Scripts**
- **Ontologies**

**Levels**

- **Morphological** (Symbols)
- **Syntactical** (Structures)
- **Semantics** (Meaning)
- **Procedural** (Inference)

# PROPOSITIONAL LOGIC

**PL** is a formal system that represents knowledge using **propositions** (simple statements) that are either **true or false**, and combines them using **logical connectives** (AND, OR, NOT, IF–THEN) to perform reasoning

**Example:**

- Ahmed is a human
- All humans are mortal
- If Ahmed is human then Ahmed is mortal

| 1. Statements | 2. Propositional symbols | Inference |
|---|---|---|
| ○ Ahmed is a human | P | Modus Ponens |
| ○ Ahmed is human then Ahmed is mortal | $P \rightarrow Q$ | $P, P \rightarrow Q \Rightarrow Q$ |
| ○ Ahmed is mortal**?** | Q | Q: Ahmed is mortal (True) |

# PROPOSITIONAL LOGIC

**Limitations**

- Cannot represent variables (e.g., "all humans")
- Must create a new symbol for every atomic proposition

- Ahmed is a human = P
- All humans are mortal **X**
- If Ahmed is human then Ahmed is mortal = $P \rightarrow Q$

# PREDICATE LOGIC

Predicate Logic is a formal logic that represents knowledge using **predicates, variables, and quantifiers** to describe objects, their properties, and relationships

- Ahmed is a human = $human(ahmed)$
- All humans are mortal = $\forall x\ human(X) \rightarrow mortal(X)$
- Ahmed is mortal**?**

- **Inference (Resolution)**
- $human(ahmed)$ (1)
- $\boldsymbol{not}\ human(X) \boldsymbol{or}\ mortal(X)$ (2) $\Rightarrow mortal(ahmed) = True$
- $\boldsymbol{not}\ mortal(ahmed)$ (3)
- $mortal(ahmed)$ (4) R(1,2) $\{X = ahmed\}$
- □ R(3,4)

# PREDICATE LOGIC

**Exercise 1:**

- Dogs are animals
- Animals will die
- Boby is a dog
- Will Boby die**?**

# PREDICATE LOGIC

**Exercise 1:**

- Dogs are animals = $\forall x\ dog(X) \rightarrow animal(X)$
- Animals will die = $\forall x\ animal(X) \rightarrow mortal(X)$
- Boby is a dog = $dog(boby)$
- Will Boby die**?**

- **Inference (Resolution)**
- $\boldsymbol{not}\ dog(X) \boldsymbol{or}\ animal(X)$ (1)
- $\boldsymbol{not}\ aniaml(Y) \boldsymbol{or}\ mortal(Y)$ (2)
- $dog(boby)$ (3) $\Rightarrow mortal(boby) = True$
- $\boldsymbol{not}\ mortal(boby)$ (4)
- $\boldsymbol{not}\ dog(Y) \boldsymbol{or}\ mortal(Y)$ (5) R(1,2) $\{X = Y\}$
- $mortal(boby)$ (6) R(3,5) $\{Y = boby\}$
- □ R(4,6)

# PREDICATE LOGIC

**Exercise 2:**

- Anyone who can read is educated
- Dolphins are not educated
- Some dolphins are intelligent
- Some intelligent cannot read**?**

# PREDICATE LOGIC

**Exercise 2:**

- Anyone who can read is educated = $\forall x\ read(X) \rightarrow educated(X)$
- Dolphins are not educated = $\forall x\ dolphin(X) \rightarrow not\ educated(X)$
- Some dolphins are intelligent = $\exists x\ dolphin(X) \boldsymbol{and}\ intelligent(X)$
- Some intelligent cannot read**?** = $\exists x\ intelligent(X) \boldsymbol{and}\ not\ read(X)$

- **Inference (Resolution)**
- $not\ read(X) \boldsymbol{or}\ educated(X)$ (1)
- $not\ dolphin(Y) \boldsymbol{or}\ not\ educated(Y)$ (2)
- $dolphin(A)$ (3)
- $intelligent(A)$ (4)
- $not\ intelligent(Z) \boldsymbol{or}\ read(Z)$ (5)
- $not\ educated(A)$ (6) R(2,3) $\{Y = A\}$
- $not\ read(A)$ (7) R(1,6) $\{X = A\}$
- $not\ intelligent(A)$ (8) R(5,7) $\{Z = A\}$
- □ R(4,8)

$\Rightarrow \exists x\ intelligent(X) \boldsymbol{and}\ not\ read(X) = True$

# PRODUCTION RULES

knowledge as **IF–THEN rules**, where conditions trigger actions or conclusions, enabling rule-based reasoning and decision-making

IF condition(s) THEN action/conclusion

- **Antecedent (IF part)**: a set of conditions that must be true
- **Consequent (THEN part)**: an action to perform or a conclusion to infer

**Example:**

**IF** a person has fever **AND** cough
**THEN** the person has flu

**Reasoning:**

- **Forward chaining:** Facts ⇒ Conclusion
- **Backward chaining:** Conclusion ⇒ Facts

# SEMANTIC NETWORKS

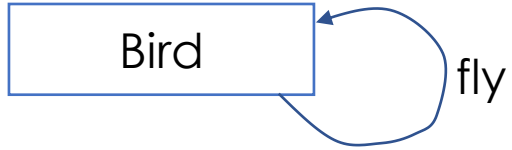
Semantic nets model knowledge as a graph, where **concepts** are represented as **nodes**, **relationships as edges**, and **properties are inherited** along hierarchical links

•Ali is a merchant


•Birds possess wings


•A merchant is a worker

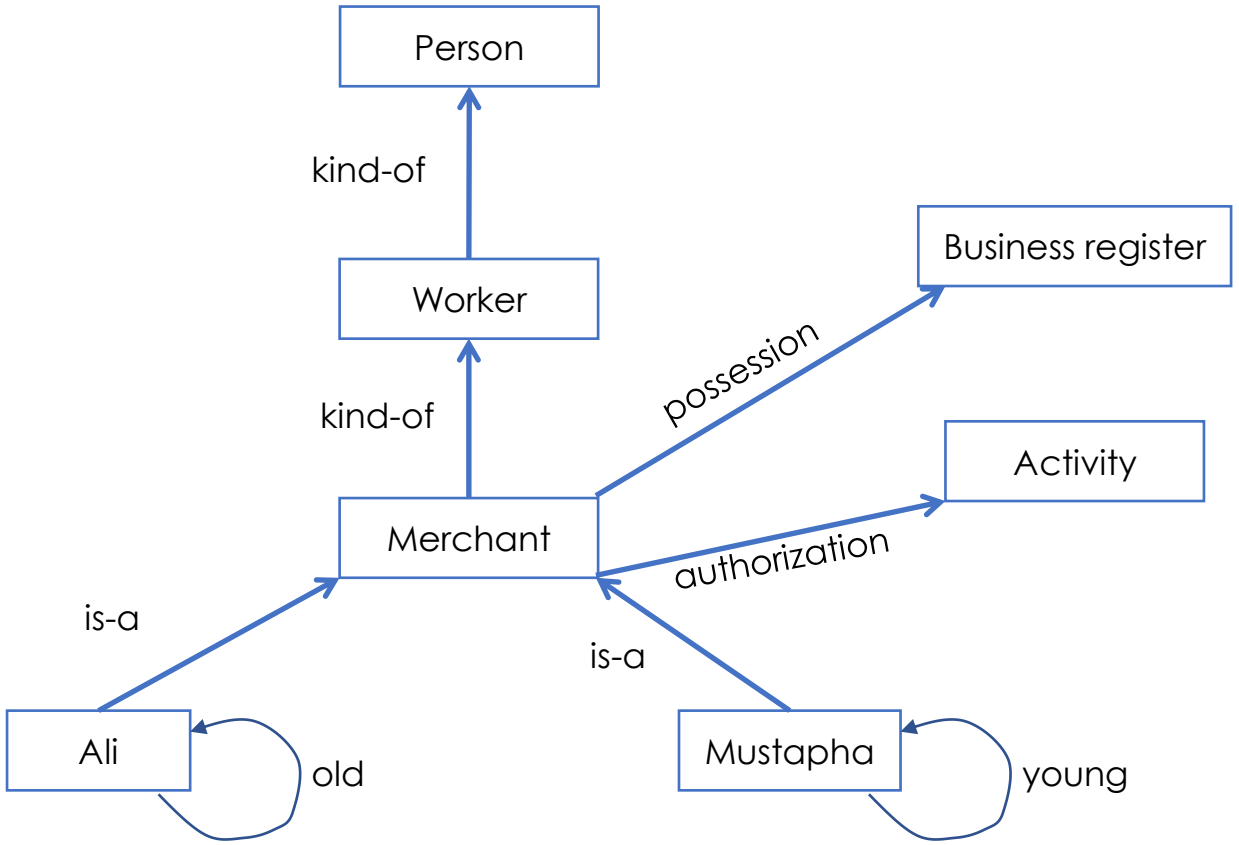

•The bird fly



# SEMANTIC NETWORKS

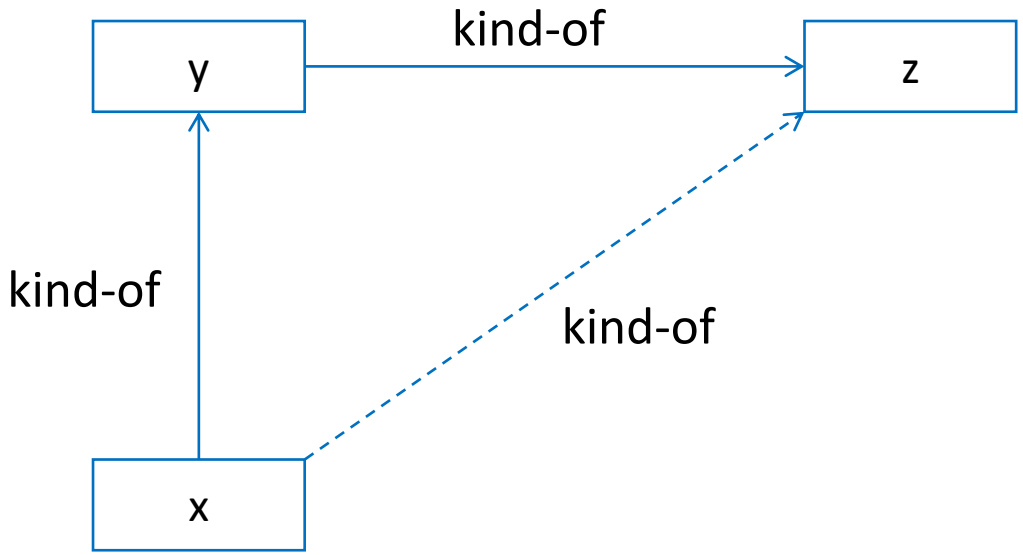
**Example:**

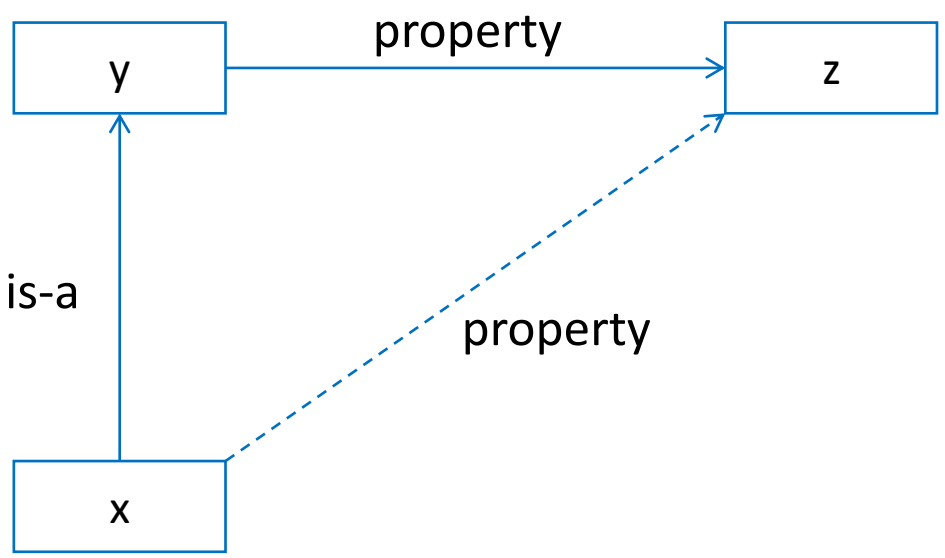
A merchant is a worker. A worker is a person. A merchant possesses a business register and a business authorization. Ali and Mustapha are two merchants. Ali is old and Mustapha is young



# SEMANTIC NETWORKS

## Deduction by Auxiliary Networks

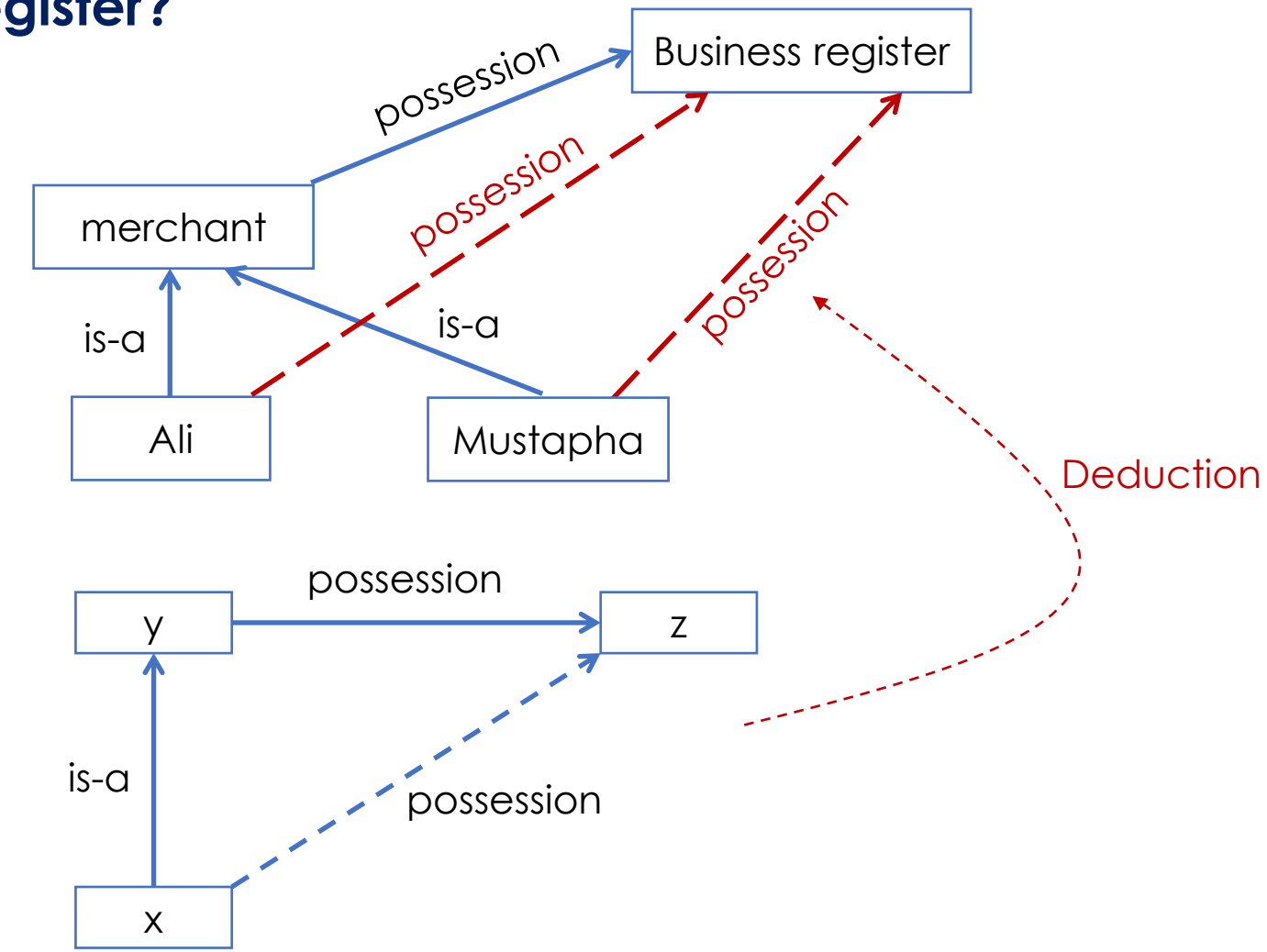


# SEMANTIC NETWORKS

## Deduction by Auxiliary Networks

**Who has a business register?**



# PROLOG

Symbolic AI programming

- Developed for symbolic AI in the 1970s.
- Built on symbolic logic (predicate calculus), not numerical computation.
- Represents knowledge as symbolic facts and rules.

# PROLOG

## OPERATORS

| | |
|---|---|
| = | Unification of right and left terms |
| == | Syntactical identity |
| =:= | Arithmetic equality |
| is | Arithmetic instanciation |
| \==, \= | Syntactical difference |
| =\= | Arithmetic difference |
| + | Addition |
| - | Substraction |
| * | Multiplication |
| / | Real division |
| // | Integer division |
| mod | Remaining of Euclidean division |
| < > =< => | Arithmetic comparison |

# PROLOG

## OPERATORS

- Unification :

```
?- X+1=2+1.
X = 2.

?- 2+X=3+1.
false.
```

- Syntactical identity:

```
?- X==X.
true.

?- dad==mom.
false.

?- "dad"=="dad".
true.

?- dad==dad.
true.
```

- Arithmetic equality:

```
?- 1+3=:=4.
true.

?- 1+3=:=3+1.
true.
```

- Arithmetic instanciation :

```
?- X is 3+4.
X = 7.

?- X is +(3,4).
X = 7.

?- X is 3*4+4*5.
X = 32.

?- X is +(*(3,4),*(4,5)).
X = 32.
```

- Syntactical difference:

```
?- dad\=mom.
true.

?- dad\==mom.
true.

?- dad\=dad.
false.
```

- Arithmetic difference:

```
?- 3=\=6.
true.

?- 3=\=3.
false.
```

# PROLOG

## PROGRAM STRUCTURE

A Prolog program consists of facts and rules (knowledge base), and queries are used to retrieve and deduce informationfrom this knowledge base.

- **Facts:** Basic assertions that are unconditionally true.

  ```
  person(ahmed).
  ```

- **Rules:** Condirional statements defining relationships.

  ```
  individual(X):-person(X).
  ```

- **Queries:** Questions asked to the Prolog system.

  ```
  ?- individual(ahmed).
  ```

# PROLOG

## SYMBOLS

| Predicate (or function, constant) | Word or letter in lowercase |
|---|---|
| Variable | Word in uppercase (or starting by capital letter or by _) |
| :- | IF<br>individual(X):-person(X). %X is an individual if X is a person |
| , | AND |
| ; | OR |
| not | Negation |
| _ | Anonymous variable(∀) |

# PROLOG

## PROPOTIONAL LOGIC

| Statements | Propositional symbols |
|---|---|
| ○ Ahmed is a human | P |
| ○ Ahmed is human then Ahmed is mortal | P → Q |
| ○ Ahmed is mortal**?** | Q**?** |

```
%Prolog program

%Facts
p.

%Rules
q:-p.

%Query
?-q.
true.
```

# PROLOG

## PREDICATE LOGIC

| Statements | Propositional symbols |
|---|---|
| ○ Ahmed is a human | $human(ahmed)$ |
| ○ All humans are mortal | $\forall x\ human(X) \rightarrow mortal(X)$ |
| ○ Is Ahmed mortal**?** | $mortal(ahmed)$**?** |

```
%Prolog program

%Facts
human(ahmed).

%Rules
mortal(X):-human(X).

%Query
?-mortal(ahmed).
true.
```

# PROLOG

## PREDICATE LOGIC

**Exercise 1:**

- Dogs are animals = $\forall x\ dog(X) \rightarrow animal(X)$
- Animals will die = $\forall x\ animal(X) \rightarrow mortal(X)$
- Boby is a dog = $dog(boby)$
- Will Boby die**?**

```
%Prolog program

%Facts
dog(boby).

%Rules
animal(X):-dog(X).
mortal(X):-animal(X).

%Query
?-mortal(boby).
true.
```

# PROLOG

## PREDICATE LOGIC

**Exercise 2:**

- Anyone who can read is educated = $\forall x\ read(X) \rightarrow educated(X)$
- Dolphins are not educated = $\forall x\ dolphin(X) \rightarrow not\ educated(X)$
- Some dolphins are intelligent = $\exists x\ dolphin(X) \boldsymbol{and}\ intelligent(X)$
- Some intelligent cannot read**?** = $\exists x\ intelligent(X) \boldsymbol{and}\ not\ read(X)$

```
%Prolog program
%Facts
dolphin(d).
intelligent(d).
%Rules
educated(X):-read(X).
unread(X):-not(read(X)).
uneducated(X):-not(educated(X)).
uneducated(X):-dolphin(X).

%Query
?- intelligent(d),unread(d).
true.
```

# PROLOG

## PRODUCTION RULES

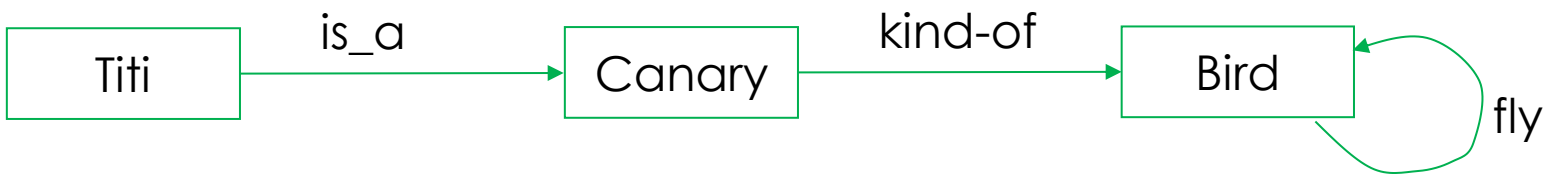
**IF** a person has fever **AND** cough
**THEN** the person has flu

```
%Prolog program

%Facts
fever(karim).
cough(karim).

%Rules
flu(X):-fever(X),cough(X).

%Query
?- flu(karim).
true.
```

# PROLOG

## Semantic Nets

A canary is a bird
A bird can fly
Titi is a canary




```
%Facts
is_a(titi,canary).
kind_of(canary, bird).
can_fly(bird).


%Rules
can_fly(X) :-is_a(X, Y),kind_of(Y,Z),can_fly(Z).

%Query
?- can_fly(titi).
true.
```

# PROLOG

## Semantic Nets

**Exercise:** Write the Prolog code for the following:

Ahmed is a student
A student is a person
Ahmed has a score of 12
A student passes if their score >=10

Will Ahmed pass**?**

```
%Facts
is_a(ahmed,student).
kind_of(student,person).
has_score(ahmed,12).

%Rules
pass(X):-is_a(X,student),has_score(X,S),S>=10.

%Query
pass(ahmed).
true.
```

# PROLOG

## Uncertainty

Each fact or rule has a confidence value in [0,1]

- 0 → completely false
- 1 → completely true
- Intermediate values → uncertainty

# PROLOG

## Uncertainty

**Predicate logic:**

Smiggle is human with confidence **0.8**

Humans are mortal with confidence **0.9**

```
%Uncertain fact
human(smiggle, 0.8).

%Uncertain rule
mortal(X,CF):-human(X,CF1), CF is CF1*0.9.

%Query
mortal(smiggle, CF).
CF = 0.72.
```

# PROLOG

## Uncertainty

**Production rule:**

**IF** fever **AND** cough

**THEN** flu (confidence = 0.7)

```
%Uncertain facts
fever(karim,0.8).
cough(karim,0.6).

%Uncertain rule
flu(X,CF):-fever(X,CF1),cough(X,CF2), Min is min(CF1,CF2), CF is Min * 0.7.

%Query
?- flu(karim,CF).
CF = 0.42.
```

# PROLOG

## Uncertainty

**Semantic net:**

A canary is a bird (0.9)

Birds can fly (0.8)

```
%Uncertain facts
kind_of(canary, bird,0.9).
can_fly(bird,0.8).

%Uncertain rule
can_fly(X,CF) :-kind_of(X, Y, CF1), can_fly(Y, CF2), CF is CF1 * CF2.

%Query
?- can_fly(canary,CF).
CF = 0.72.
```

# PROLOG

Properties

- Prolog is **correct**:
  - It only provides logically correct answers.
- Prolog is **complete**:
  - When it stops, we can be sure that it has provided all the necessary answers.

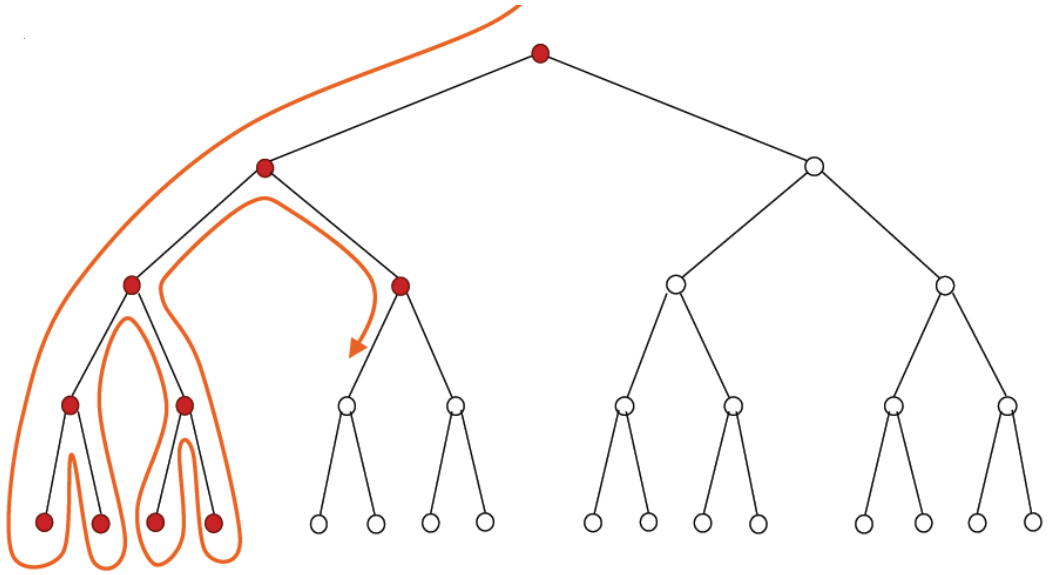
# PROLOG

Properties

**Reasoning principle:**

Prolog use a systematic search using trees. Each node is a list of goals, and the traversal is according to the technique:

**Left-to-Right, Depth-first, with Backtracking.**



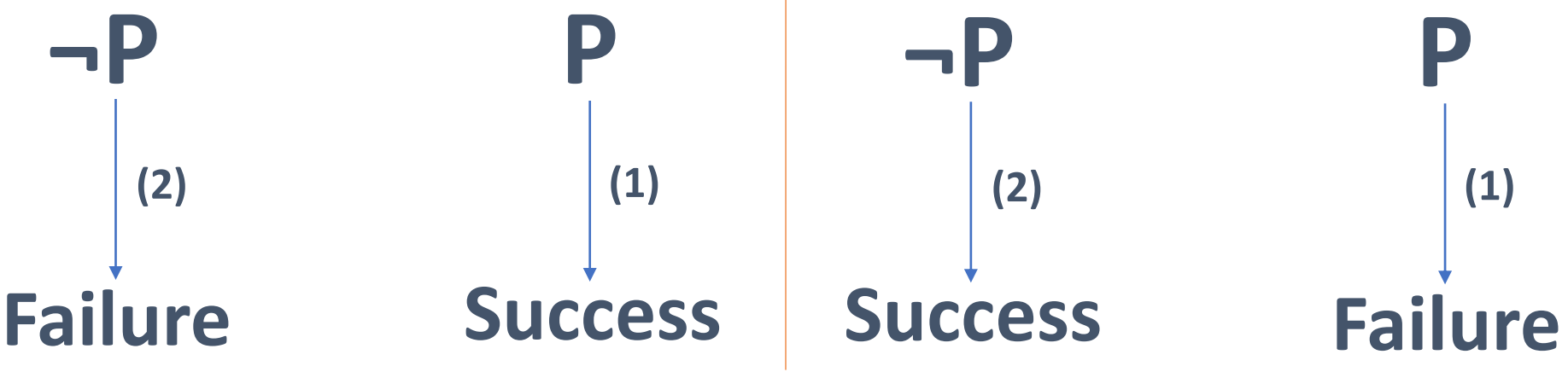
# PROLOG

Properties

## Negation processing:

When encountering negations, Prolog reasons with the principle of **Negation by Failure**.

- If Prolog reasoning applied to goal **P** yields success, then the goal **¬P** is considered to result in failure.
- If Prolog reasoning applied to goal **P** yields failure, then the goal **¬P** is considered to result in success.



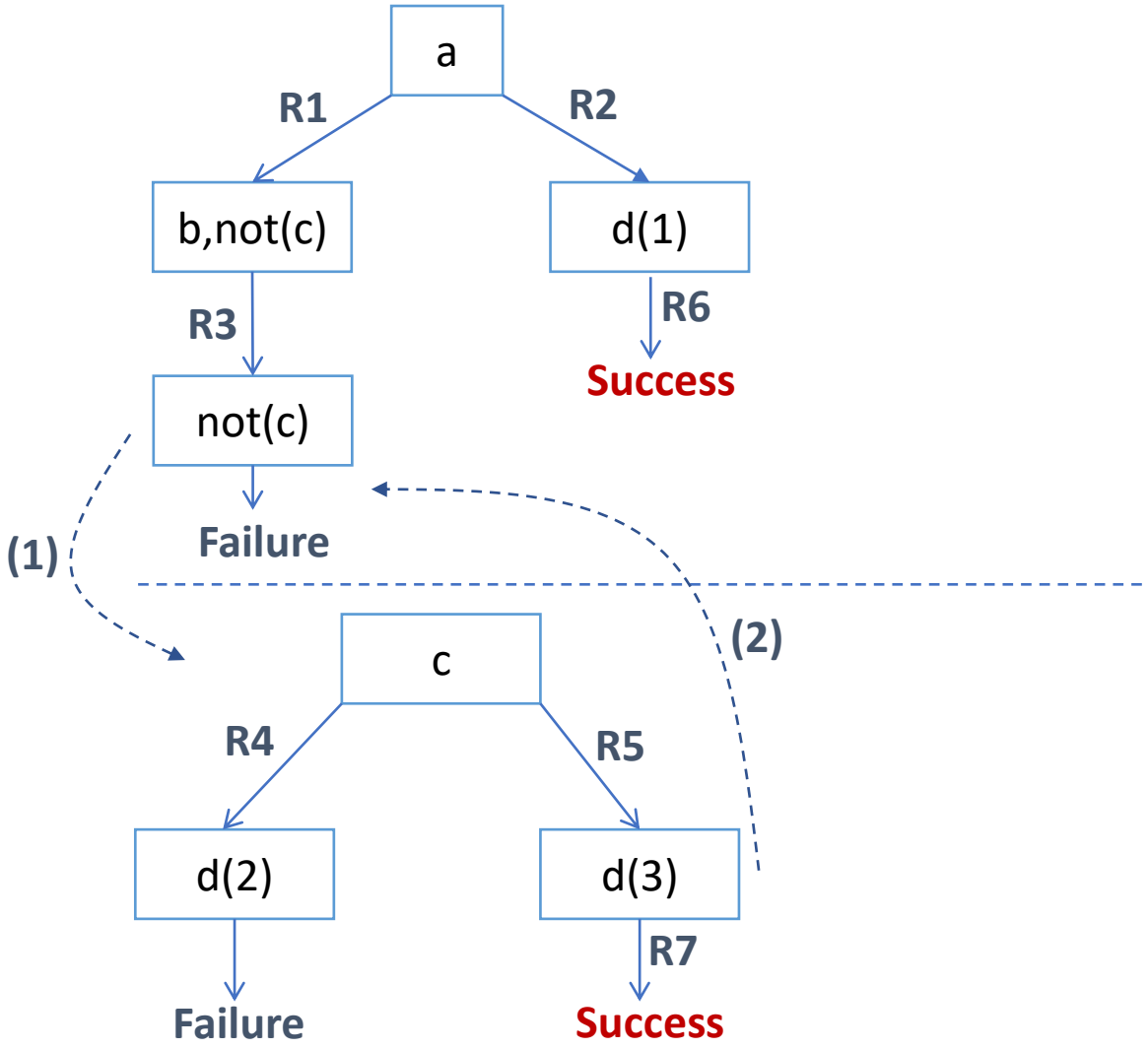
# PROLOG

Negation by failure

## Example 1:

R1: a:-b,not(c).
R2: a:-d(1).
R3: b.
R4: c:-d(2).
R5: c:-d(3).
R6: d(1).
R7: d(3).

Query: a ?



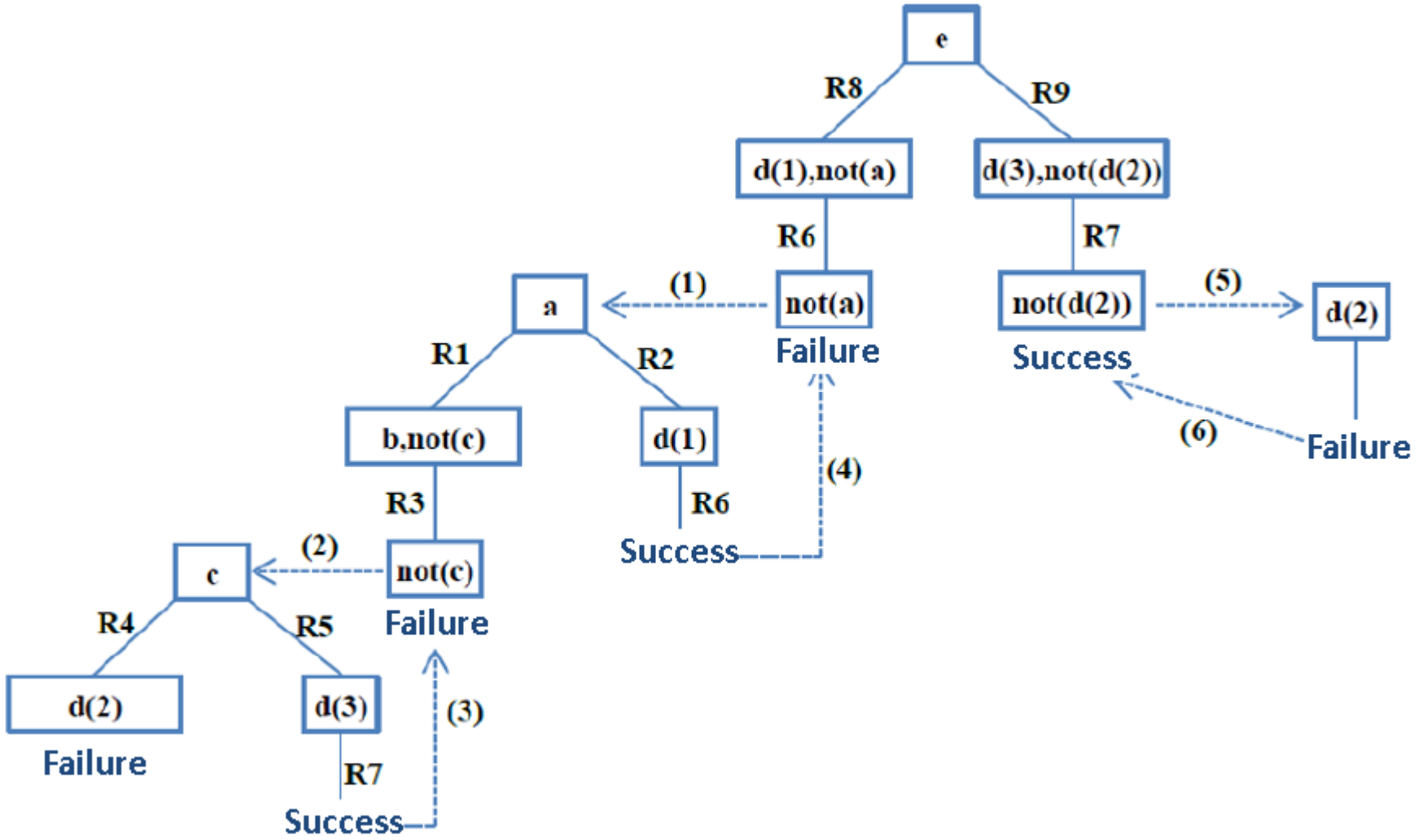
# PROLOG

Negation by failure

## Example 2:

R1: a:-b,not(c).
R2: a:-d(1).
R3: b.
R4: c:-d(2).
R5: c:-d(3).
R6: d(1).
R7: d(3).
R8: e:-d(1),not(a).
R9: e:-not(d(2)),d(3).

Query: e ?



# PROLOG

## The CUT (!)

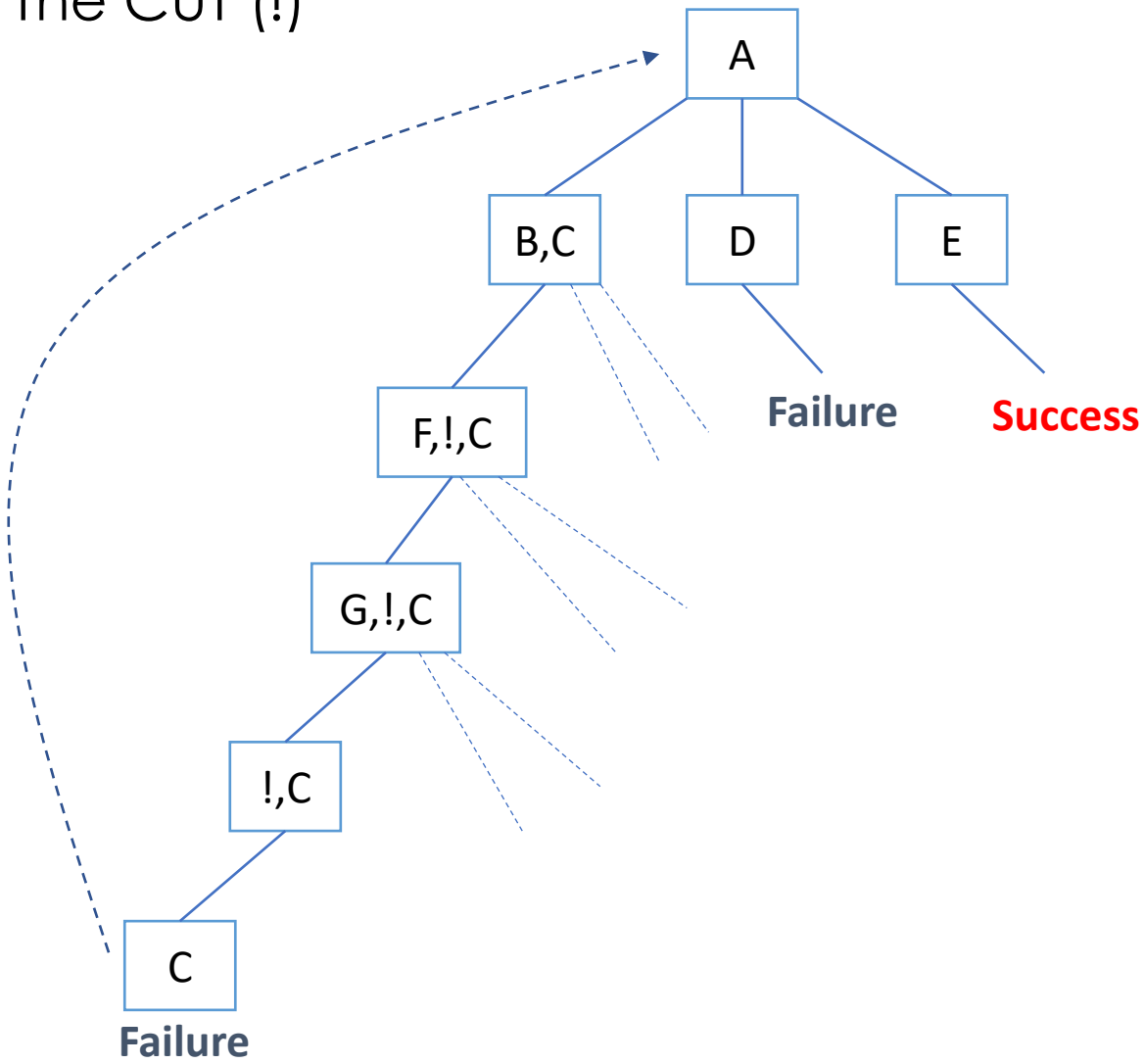
- The **cut "!"** is a very special predicate; it always succeeds (true) and it affects the way the reasoning tree is traversed.
- During the traversal of the reasoning tree, if a list of goals starting with a cut "!" is encountered:
  - It is necessary to backtrack directly above the goal that introduced this cut.
  - When the cut is encountered, it prunes all choices originating from the goal that introduced the cut.

# PROLOG

The CUT (!)

**Example:** A?

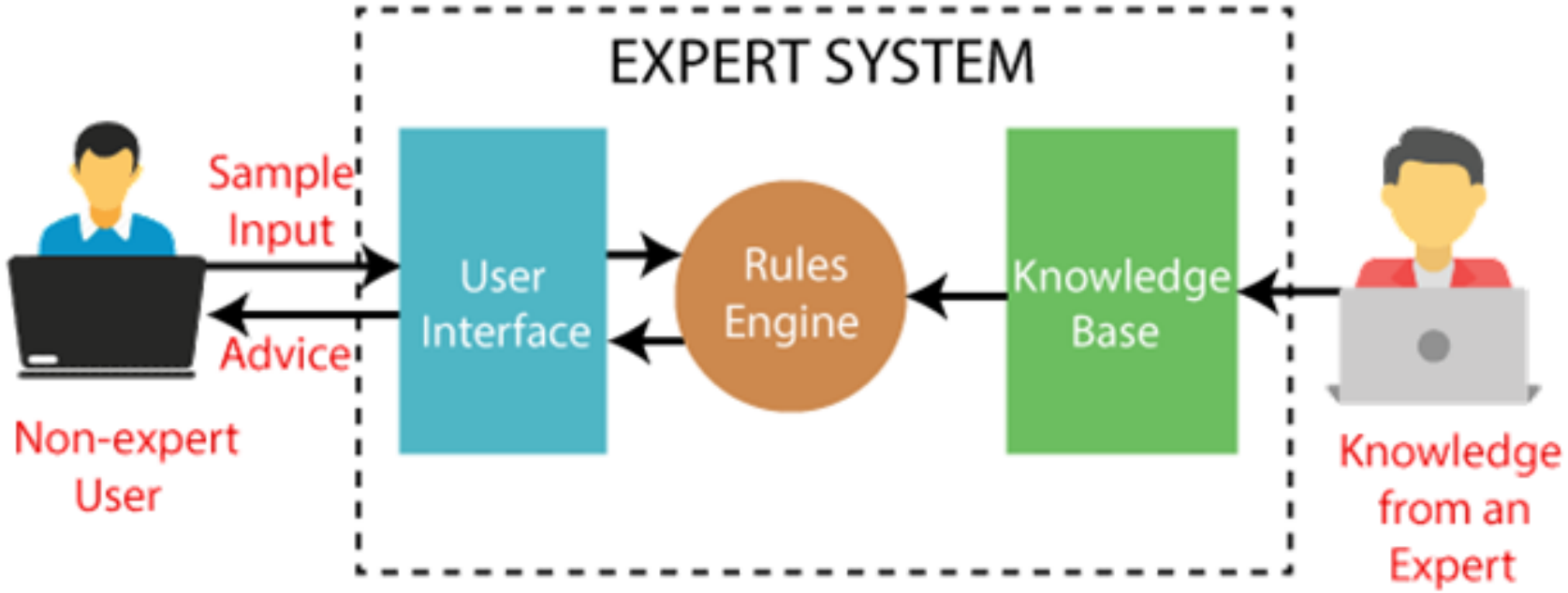
A:-B,C.
A:-D.
A:-E.
B:-F,!.
B:-H.
B:-I.
E.
F:-G.
F:-J.
F:-K.
G.
G:-L.
G:-M.



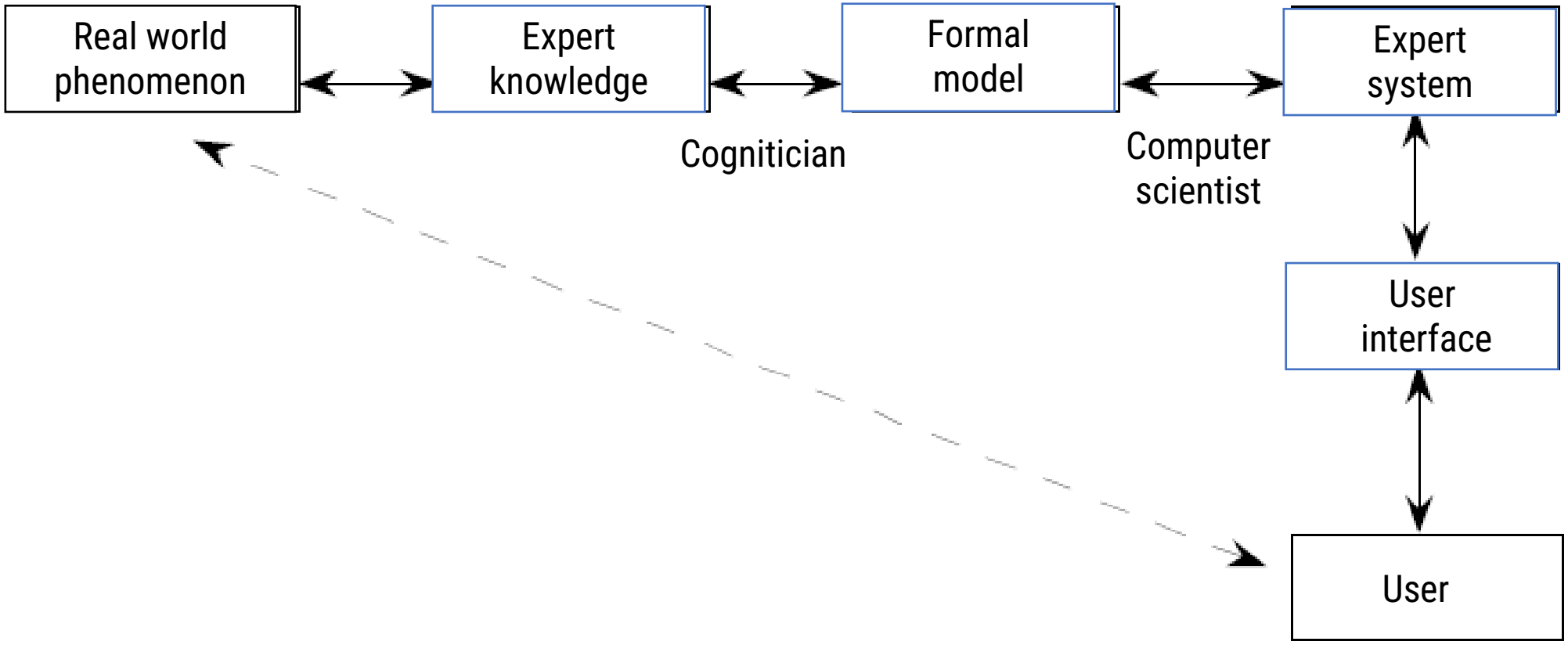
# EXPERT SYSTEMS

# EXPERT SYSTEM ?

An expert system aims to model the behavior of a human expert performing a task to solve  a problem for which there is no algorithm, within a very specific domain.



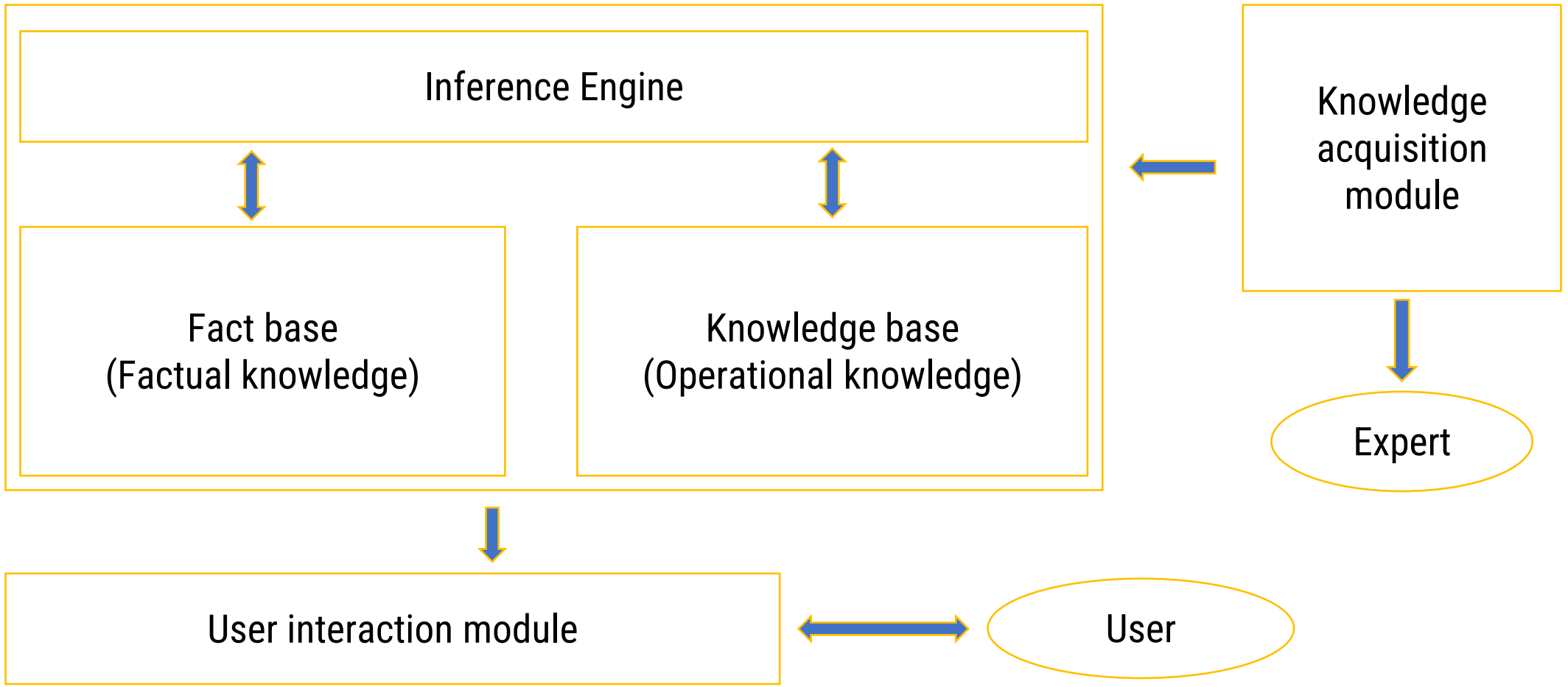
# EXPERT SYSTEM ?



# Expert System Applications

- **Computer Science:** maintenance assistance, programming,..
- **Medicine:** diagnostic assistance, patient monitoring,..
- **Finance and banking:** risk assessment,..
- **Industry:** diagnostics

# EXPERT SYSTEM ARCHITECTURE



# EXPERT SYSTEM ARCHITECTURE

## 1.Fact Base

- Represents the working memory of the system.
- Contains data specific to the problem being addressed (assertions describing situations considered as established or to be established)
- It stores intermediate results (trace of the reasoning).
- The fact base is updated as reasoning progresses

# EXPERT SYSTEM ARCHITECTURE

## 2. Knowledge Base

- Gathers information specific to the field of expertise. This information is provided directly by the expert or accumulated by the system over the experiments

- The knowledge base is written in a knowledge representation language (Example: production rules, referred to as a rule base).

# EXPERT SYSTEM ARCHITECTURE

## 3.Knowledge Acquisition Module:

Provides the expert with the ability to transmit their knowledge to the system in order to build the knowledge base (KB).

## 4.User Interaction Module:

Allows the user to query the expert system in order to:

- Solve their own problems.
- Acquire expertise similar to that of the expert.

# EXPERT SYSTEM ARCHITECTURE

## 5.Inference Engine:

This is a program that utilizes the elements of the knowledge base (KB) and the fact base (FB) to perform reasoning. It is characterized by:

- A) A basic cycle.
- B) A search strategy.
- C) A method of chaining (forward or backward)

# Inference Engine

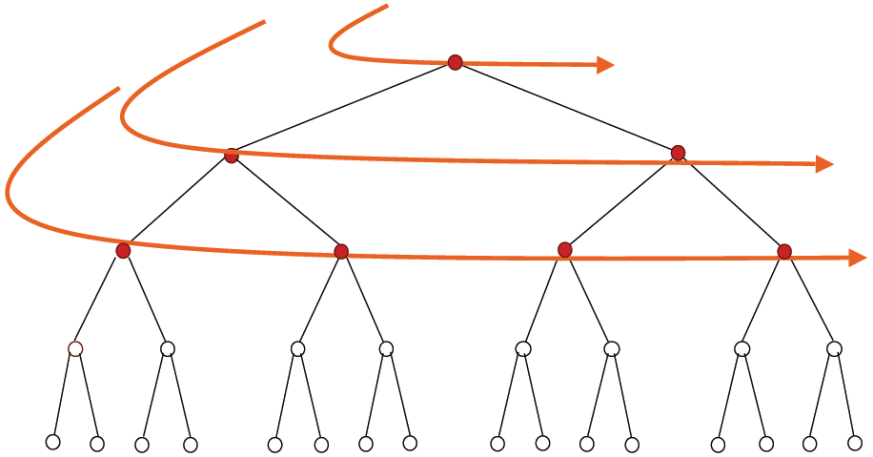
**A. Basic Cycle:**

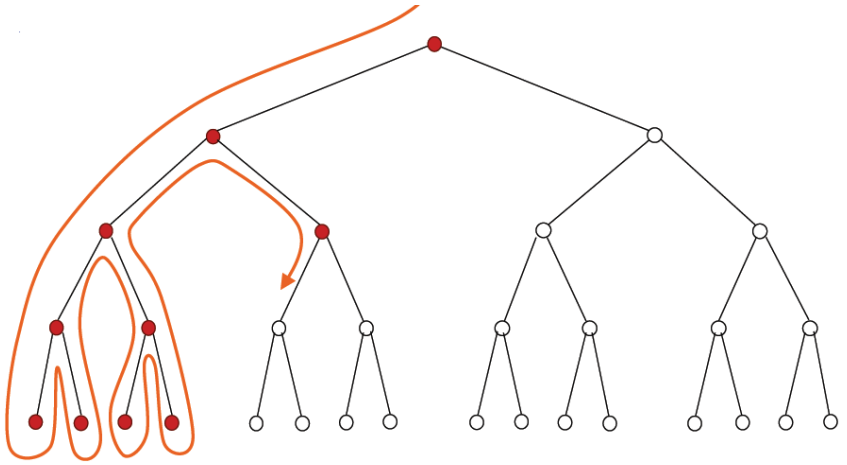
- **Selection Phase:** Its purpose is to sort and gather into a subset those rules from the knowledge base that deserve more attention than others.
- **Filtering Phase:** Determines the set of applicable rules based on the results of the first phase.
- **Conflict Resolution Phase:** Characterized by the choice of the rule to apply according to a specific strategy:
  - The first rule in appearance.
  - The most reliable rule (for example, based on likelihood coefficients).
- **Execution Phase:** Involves activating the chosen rule from the previous step. This action allows one or more new facts to be added to the fact base.

# Inference Engine

**B. Search Strategies:**

- **Breadth-first:** Explores all the nodes at the present depth prior to moving on to nodes at the next depth level.
- **Depth-first:** Explores as far as possible along each branch before backtracking.
- **Heuristic:** Utilizes methods to speed up the process of finding a satisfactory solution, often through rules of thumb that prioritize paths that seem likely to lead to a goal.





# Inference Engine

**C. Chaining Methods (Reasoning Strategy):**

- **Forward Chaining (Deductive Reasoning):**
  - •Starting from the facts provided by the user, the inference engine deduces conclusions.
  - •Use the newly obtained facts to trigger other rules.
  - •Stop the reasoning process when no goal can be deduced.
  - •This reasoning mode is used when we do not have a precise idea of the goal to be achieved.
- **Backward Chaining (Inductive Reasoning):**
  - •Starting from the conclusions, the inference engine tries to verify the truth of the premises.
  - •If the premises exist in the fact base, then the problem is solved.
  - •Otherwise, the unverified conditions become sub-goals to prove.
- **Mixed Chaining:**
  - •Combine forward and backward chaining methods as needed.

# Chaining

**Consider the following Knowledge Base:**

**Rule 1:** If the person has a diploma

And the person has experience

Then the person meets all the conditions required by the company.

**Rule 2:** If the person meets all the conditions required by the company

And the company offers a job

Then the person is recruited.

**Rule 3:** If the company offers a job

Then there is a budgeted position.

**Rule 4:** If the person is recruited

Then the person receives a salary

And the person is not unemployed.

**Consider the following Data Base:**

The company offers a job

The person has a diploma

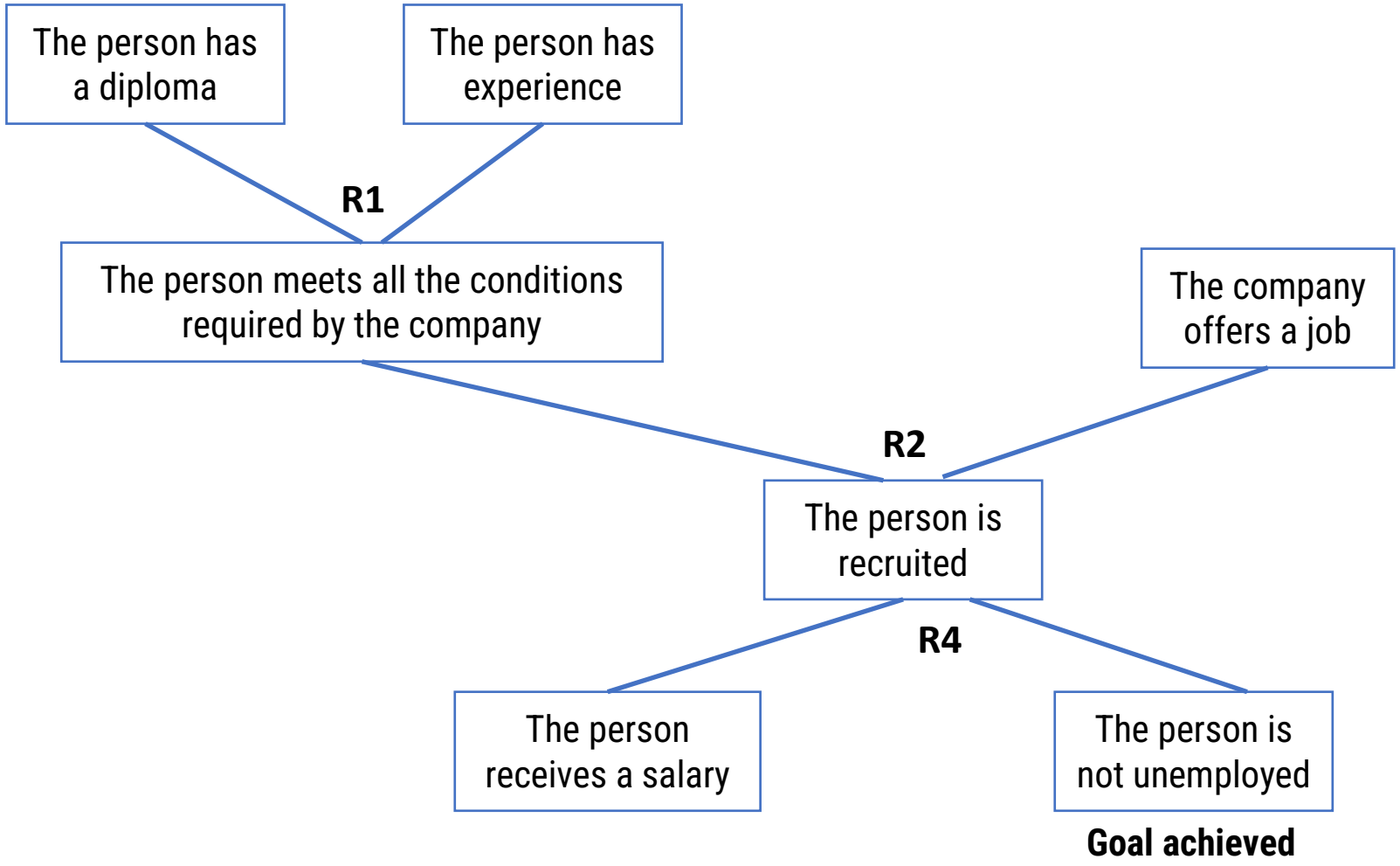
The person has experience

**Goal:** The person is not unemployed **?**

# CHAINING

**Example:** Forward chaining (From the facts, the inference engine deduces all possible facts)

**Question:** Is the person not unemployed?




**FB:**

The company offers a job

The person has a diploma

The person has experience

**KB:**

**Rule 1:** If the person has a diploma

And the person has experience

Then the person meets all the conditions required by the company.

**Rule 2:** If the person meets all the conditions required by the company

And the company offers a job

Then the person is recruited.

**Rule 3:** If the company offers a job

Then there is a budgeted position.

**Rule 4:** If the person is recruited

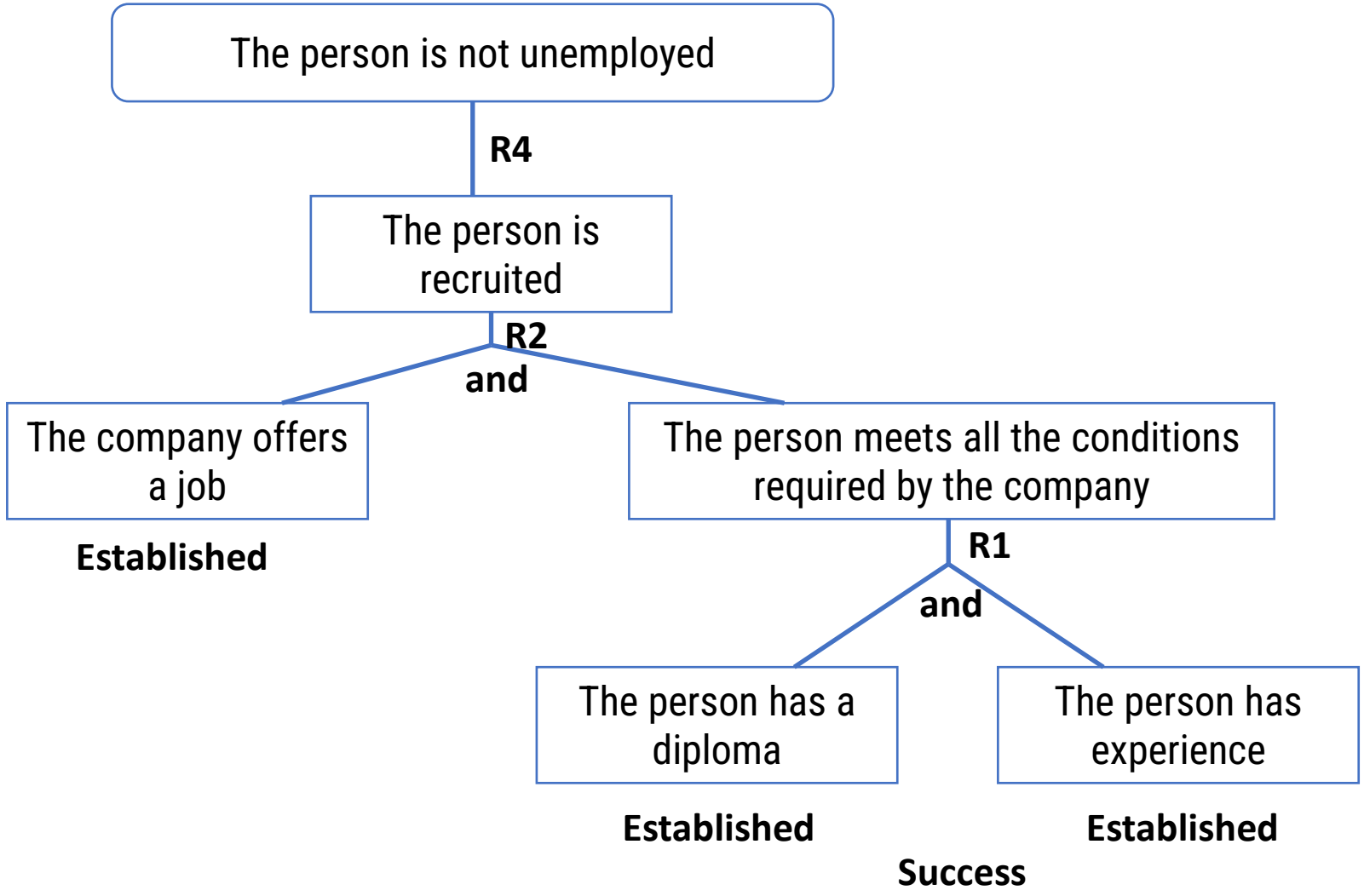
Then the person receives a salary

And the person is not unemployed.

# CHAINING

**Example : Backward chaining** (The inference engine tries to verify a goal)

Goal: The person is not unemployed**?**




**FB:**

The company offers a job

The person has a diploma

The person has experience

**KB:**

**Rule 1:** If the person has a diploma

And the person has experience

Then the person meets all the conditions required by the company.

**Rule 2:** If the person meets all the conditions required by the company

And the company offers a job

Then the person is recruited.

**Rule 3:** If the company offers a job

Then there is a budgeted position.

**Rule 4:** If the person is recruited

Then the person receives a salary

And the person is not unemployed.

# CHAINING

**Exercise:** Deduce the goal X using a forward chainng and then a backward chainng:

**Rules:**
R1 : R,F,N $\rightarrow$ D
R2 : F,G $\rightarrow$ A
R3 : S,D $\rightarrow$ A
R4 : R $\rightarrow$ L
R5 : F $\rightarrow$ N
R6 : A,L $\rightarrow$ X
R7 : S $\rightarrow$ F
R8 : L,S $\rightarrow$ A

**Facts:**
R,S

**Goal:** X ?

**Conflict resolution criterion:** Retain the first rule appearing

# CHAINING

**Forward chaining:** Deduce X?

**Rules:**

R1 : R,F,N → D
R2 : F,G → A
R3 : S,D → A
R4 : R → L
R5 : F → N
R6 : A,L → X
R7 : S → F
R8 : L,S → A

**Facts:**

R,S

| (Facts) | (Applicable rules) |
|---|---|
| R,S | (R4, R7) |
| ↓ R4 | |
| R,S,L | (R7, R8) |
| ↓ R7 | |
| R,S,L,F | (R5, R8) |
| ↓ R5 | |
| R,S,L,F,N | (R1, R8) |
| ↓ R1 | |
| R,S,L,F,N,D | (R3, R8) |
| ↓ R3 | |
| R,S,L,F,N,D,A | (R6, R8) |
| ↓ R6 | |
| R,S,L,F,N,D,A,X | |

**Goal achieved**

# CHAINING

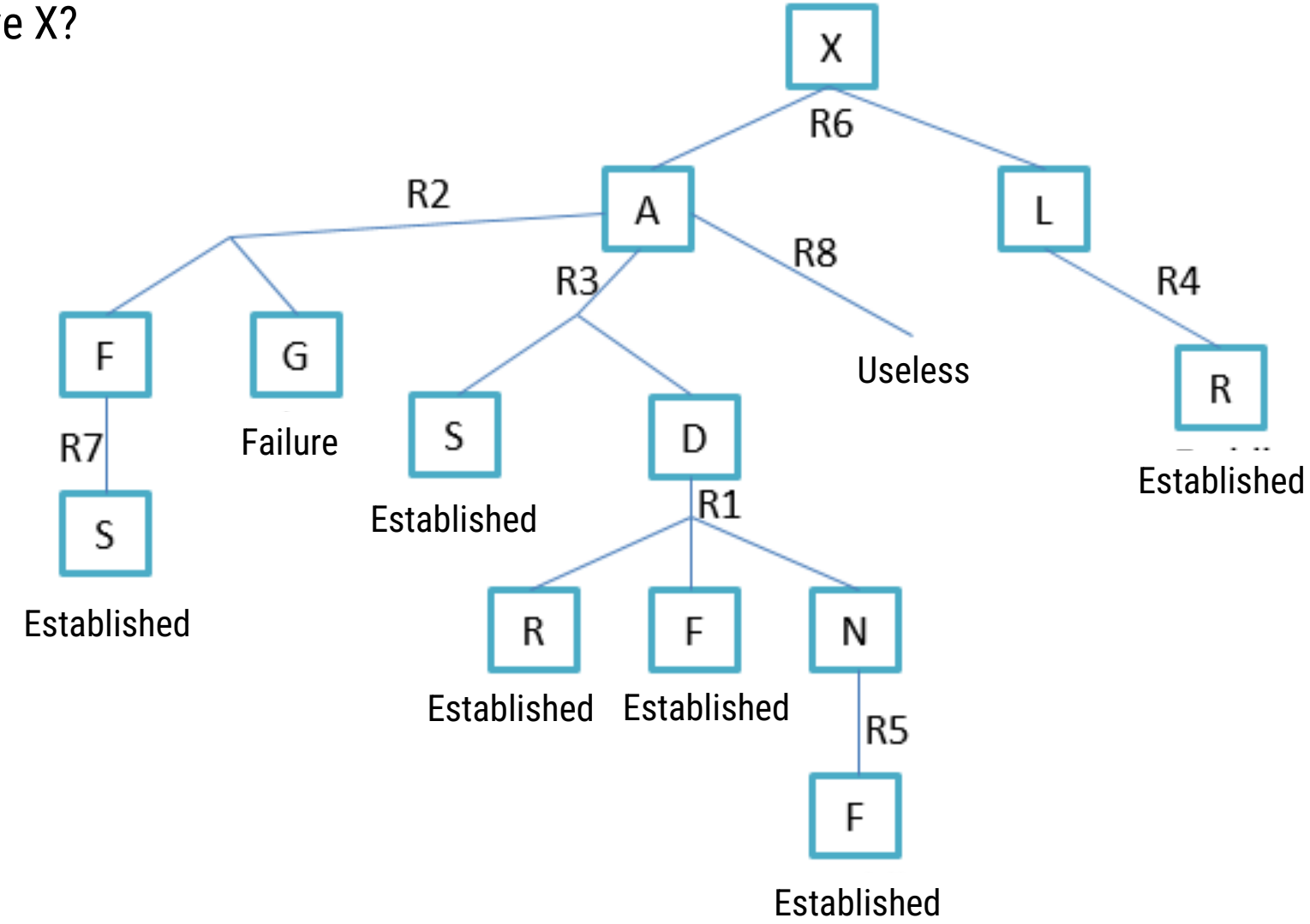
**Backward chaining:** Prove X?

**Rules:**

R1 : R,F,N → D
R2 : F,G → A
R3 : S,D → A
R4 : R → L
R5 : F → N
R6 : A,L → X
R7 : S → F
R8 : L,S → A

**Facts:**

R,S



# CHAINING

Prove H using forward and backward chaining?:

**Rules:**
R1. B,D,E $\rightarrow$ F
R2. D,G $\rightarrow$ A
R3. C,F $\rightarrow$ A
R4. C $\rightarrow$ D
R5. D $\rightarrow$ E
R6. A $\rightarrow$ H
R7. B $\rightarrow$ X
R8. X,C $\rightarrow$ A

**Facts:**
B,C

**Conflict resolution criterion:** Retain the first rule appearing

# Graph Pathfinding vs Rule-base ES

| Pathfinding in a graph | Functioning of a rule-based expert system |
|---|---|
| Node | Fact |
| Arc | Rule |
| Network | Rule base |
| Node queue | Fact base |
| Starting | Hypothesis (fact to prove) |
| Destination | Conclusion |
| Vehicle | Inference engine |
| Identify accessible nodes | Select applicable rules |
| Reach a node | Apply a rule |

# EXPERT SYSTEMS

EXAMPLES

## ❑ DENDRAL

- Developed by Feigenbaum in 1965.
- Written in LISP.
- Objective: To identify a chemical structure from chemical, physical, and spectrometric measurement results.

**Principle:** Deduce from measurement results (all possible information)

- Ask the user if any information is missing
- Synthesize and conclude

**Disadvantages:** Programmed in a classical manner (difficult to maintain) No possibility of explanation

# EXPERT SYSTEMS

## EXAMPLES

### ❑ MYCIN

- Developed at Stanford in 1974.
- Written in LISP.
- Objective: Diagnosis and treatment of certain blood diseases (infections).

MYCIN represents its knowledge in the form of IF-THEN rules.

**Example of rule:**

*IF the infection is primary-bacteremia*

*AND the site of the culture is one of the sterile sites*

*AND the suspected portal of entry is the gastrointestinal tract*

*THEN there is suggestive evidence (0.7) that infection is bacteroid.*

- MYCIN Goal-driven system (backward chaining)
- New versions of MYCIN were subsequently developed: EMYCIN, NEOMYCIN