DRONACHARYA
College of Engineering

INTELLIGENT SYSTEMS (CSE-303-F)

Section A

PROLOG

Outline

- What is Prolog?
- Language syntax
- Rules
- Questions
- Backtracking
- Conclusions

What Is Prolog?

- Logic programming is a programming language paradigm in which logical assertions are viewed as a program
- Prolog is described as a series of logical assertions or it is a logic-based language
- With a few simple rules, information can be analyzed.

Representation in logic

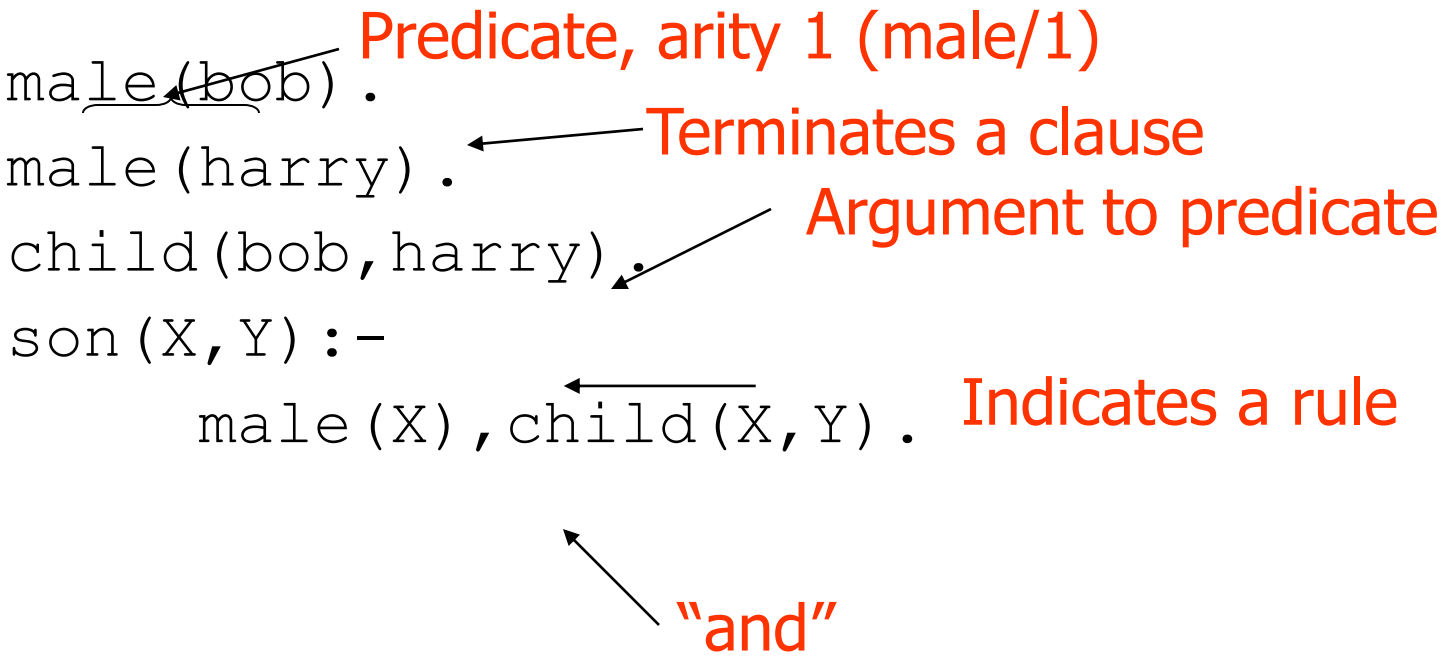
- ∀ x : pet(x) Λ small (x) → apartment(x)
- ∀ x : cat(x) V dog(x) → pet(x)
- ∀ x : poodle(x) → dog(x) Λ small(x)
- Poodle(abs)

Representation in PROLOG

- Apartment (x) :- pet(x), small(x)
- Pet (x) :- dog (x)
- Dog (x) :- poodle (x)
- Small(x) :- poodle (x)
- Poodle(abs)

Syntax

- .pl files contain lists of clauses
- Clauses can be either facts or rules



Rules

- Rules combine facts to increase knowledge of the system

```
son(X,Y):-
     male(X),child(X,Y).
```

- X is a son of Y if X is male and X is a child of Y

Questions

- In Prolog the queries are statements called directive
- Syntactically, directives are clauses with an empty left-hand side.

Example : ? - grandparent(X, W).

This query is interpreted as : *Who is a grandparent of X ?*

- The result of executing a query is either *success or failure*
- Success, means the goals specified in the query holds according to the facts and rules of the program.
- Failure, means the goals specified in the query does not hold according to the facts and rules of the program

- Ask the Prolog virtual machine questions
- Composed at the ?- prompt
- Returns values of bound variables and yes or no

```
?- son(bob, harry).
yes
?- king(bob, france).
no
```

Questions [cont'd]

- Can bind answers to questions to variables
- Who is bob the son of? (X=harry)

```
?- son(bob, X).
```

- Who is male? (X=bob, harry)

```
?- male(X).
```

- Is bob the son of someone? (yes)

```
?- son(bob, _).
```

- No variables bound in this case!

Backtracking

- How are questions resolved?

?- son(X,harry).

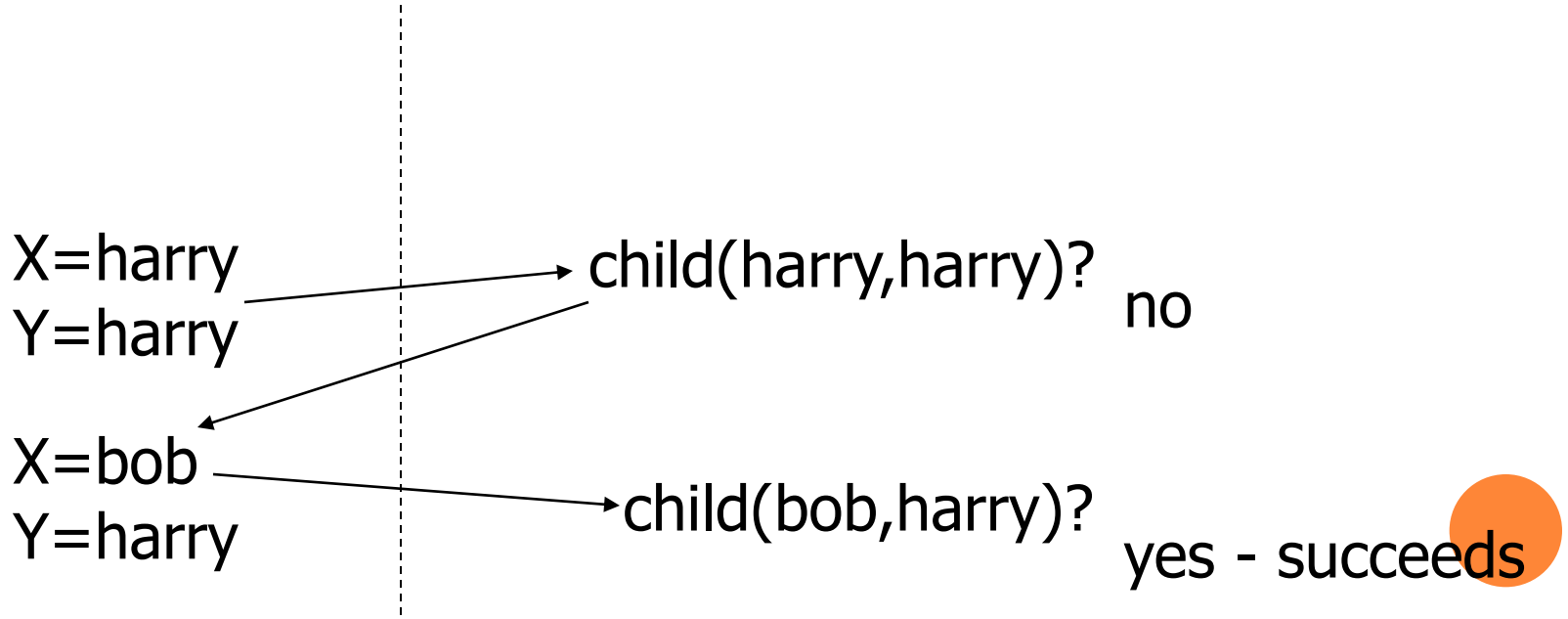
- Recall the rule:

```
son(X,Y):-
     male(X),child(X,Y).
```

Backtracking [cont'd]

- Y is bound to the atom "harry" by the question.

male(X) child(X,Y)



Applications

- Intelligent systems
- Complicated knowledge databases
- Natural language processing
- Logic data analysis

Conclusions

Strengths:

- Strong ties to formal logic
- Many algorithms become trivially simple to implement

Weaknesses:

- Complicated syntax
- Difficult to understand programs at first sight

Issues

- What applications can Prolog excel at?
- Is Prolog suited for large applications?
- Would binding the Prolog engine to another language be a good idea?

Forward v/s Backward Chaining:

- Infer means " to derive as a conclusion from facts or premises".
  There are 2 common rules for deriving new facts from rules and known facts.These are
  Modus Ponens and Modus Tollens.

- **7.1.1 MODUS PONENS**
- *most common inference strategy
  *simple ,reaoning based on it is easily understood.

  The rule states that when A is known to be true and if a rule states " If A then B "
  it is valid to conclude that B is true.

- **7.1.2 MODUS TOLLENS**
- When B is false, rule If A, then B
  then A is false.
- E.g: Rule : IF Ahmet's CAR IS DIRTY
  THEN Ahmet HAS BEEN DRIVING OUTSIDE ANKARA
- Given fact : Ahmet has not been outside Ankara.
  New rule : Ahmet car is not dirty.
- This conclusion seems quite obvious but cannot be reached by most expert systems. Because they use modus ponens for deriving new facts from rules.

- Inference engine performs 2 major tasks:

1) examines existing facts and rules and adds new facts when possible

2) decides the order in which inferences are made.

Two referencing methods:

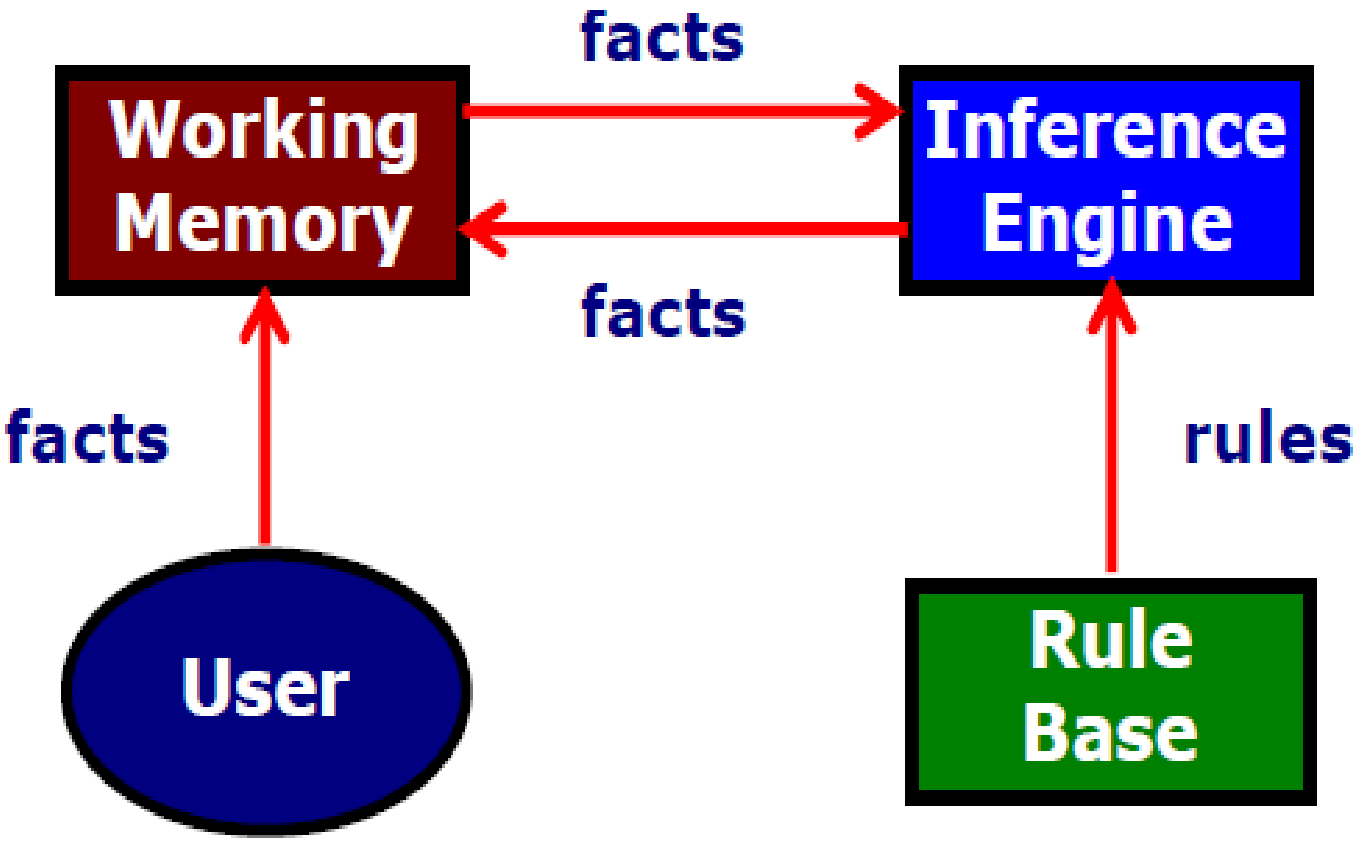
- Forward Chaining
- Backward chaining

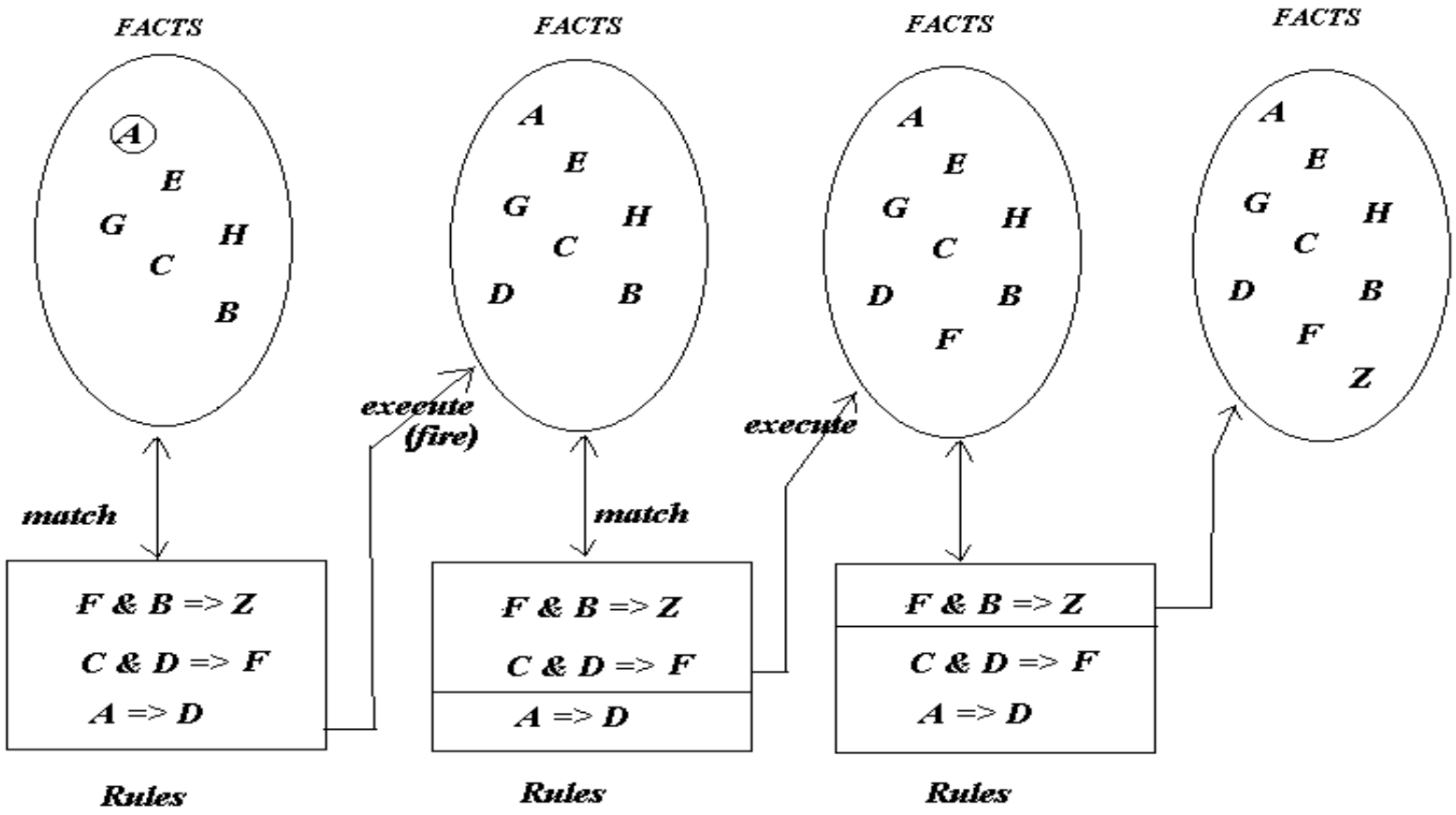
- **Forward chaining : also called data driven.**

It starts with the facts, and sees what rules apply.

- **Backward chaining : also called goal driven.**

It starts with something to find out, and looks for rules that will help in answering it.


facts
Working Memory
Inference Engine
facts
facts
rules
User
Rule Base



Problem: Does situation Z exists or not ?

Forward chaining Algorithm

Given m facts $F_1, F_2, ..., F_m$? N RULES $R_1, R_2, ... R_n$

repeat for i ?- 1 to n do

if one or more current facts match the antecedent of Ri then

1) add the new fact(s) define by the consequent

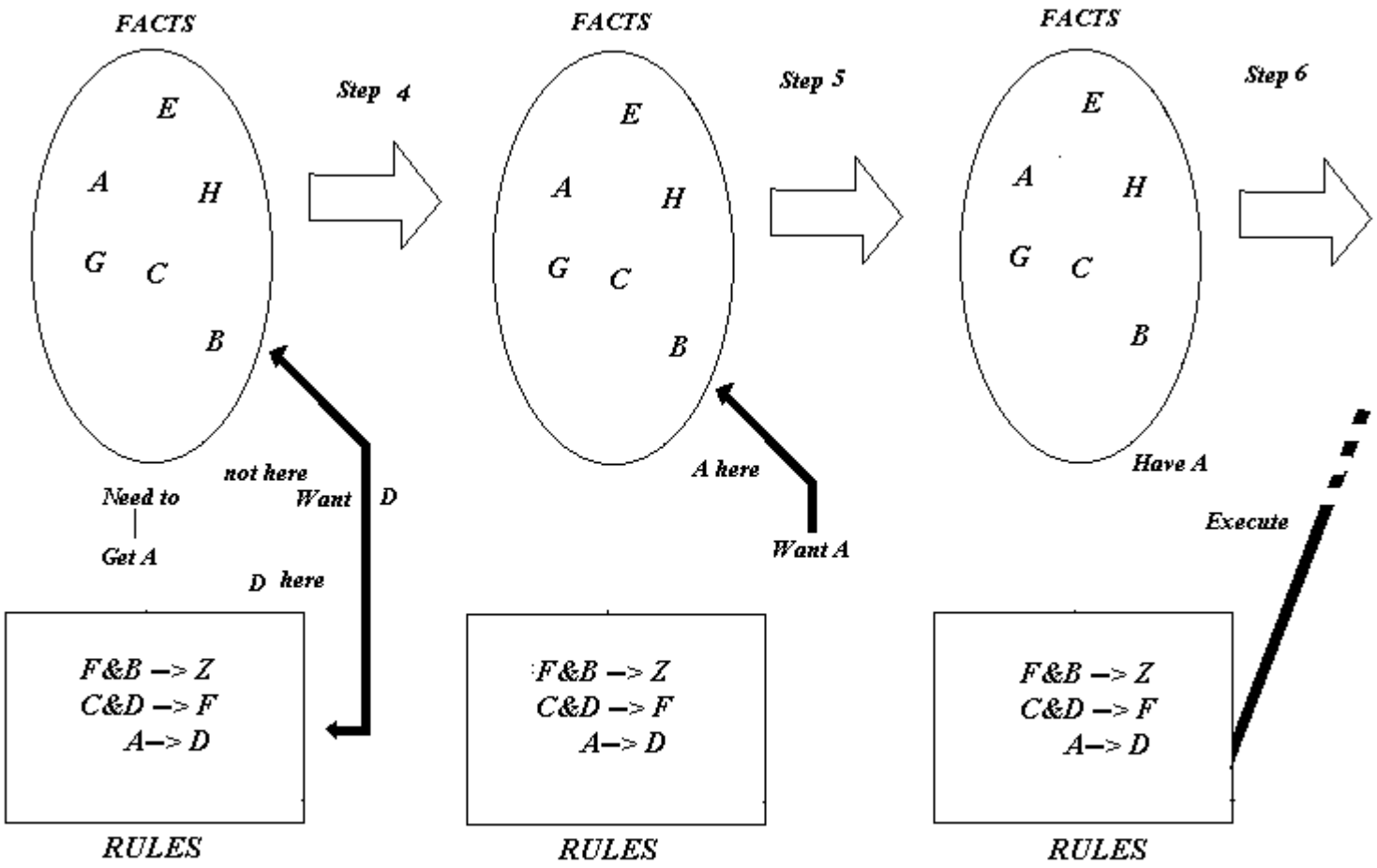
2) flag the rule that has been fired

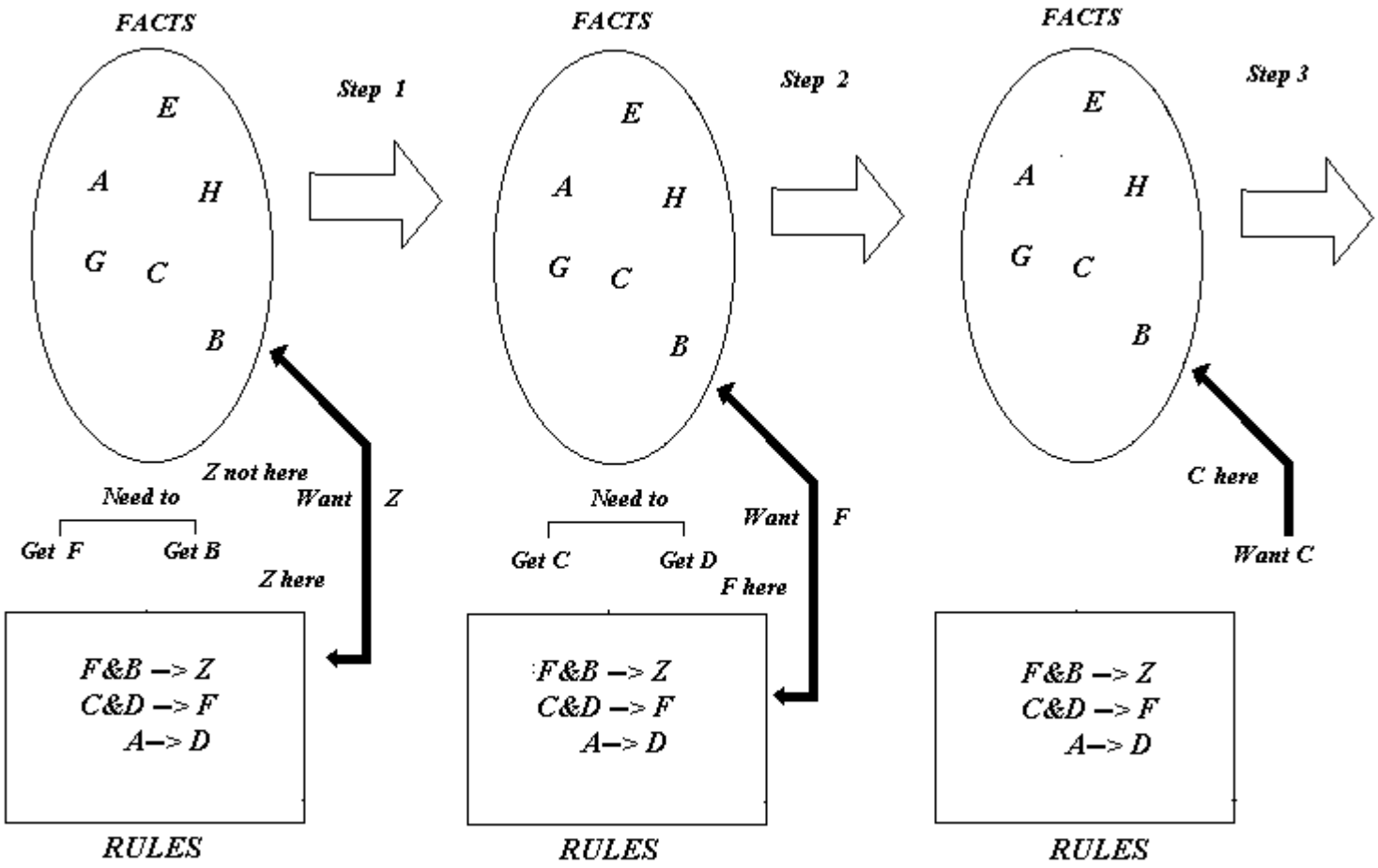
3) increase m until no new facts have been produced.

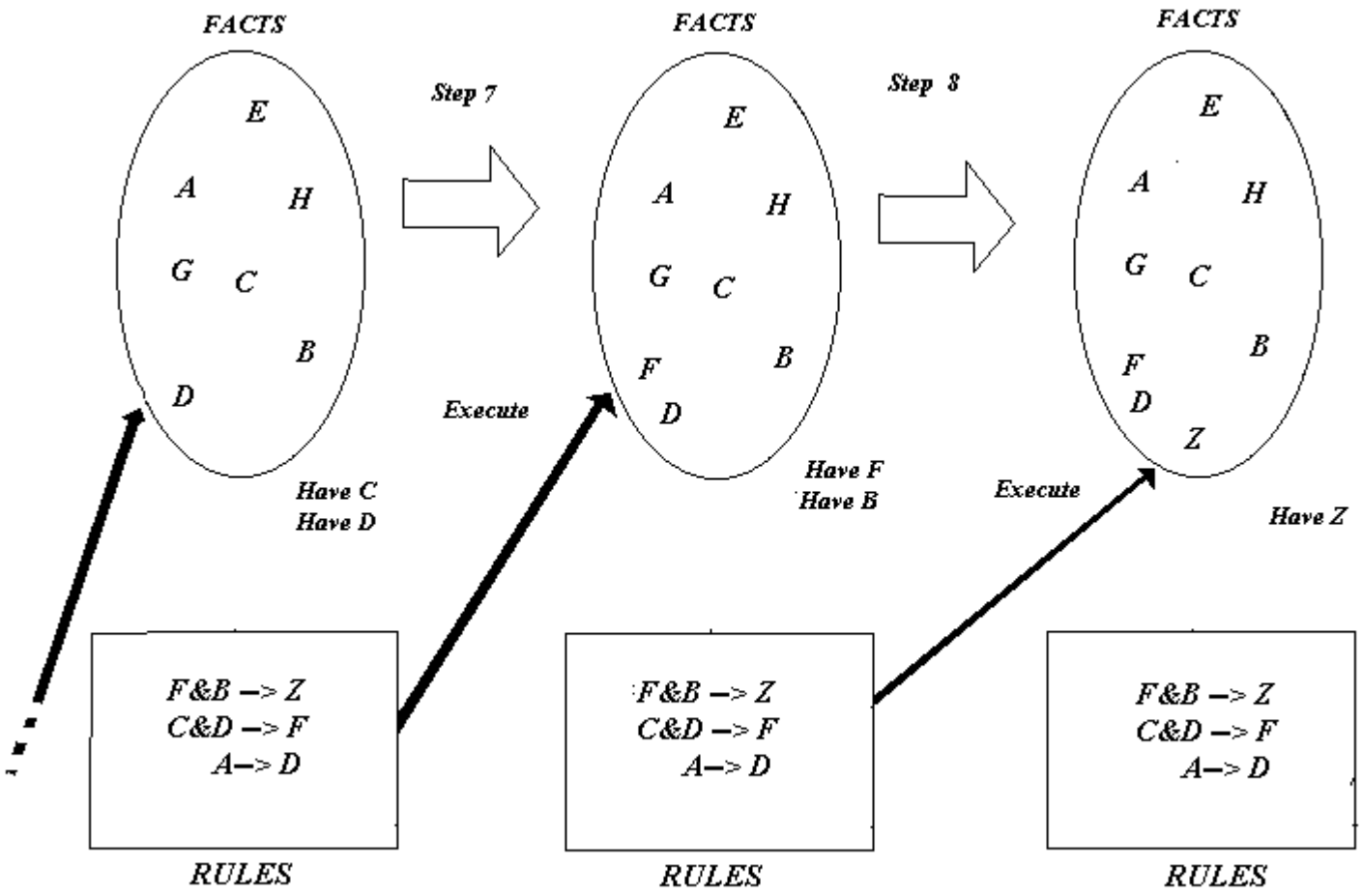
until no new facts have been produced.

Backward Chaining:

- With this inference method the system starts with what it wants to prove, e.g.,that situation Z exists, and only executes rules that are relavent to establishing it.
  Figure following shows how bacward chaining would work using the rules from the forward chaining example.


FACTS
E
A
H
G
C
B
Step 4
not here
Need to
Want
D
Get A
D here
F&B –> Z
C&D –> F
A–> D
RULES
FACTS
E
A
H
G
C
B
Step 5
A here
Want A
F&B –> Z
C&D –> F
A–> D
RULES
FACTS
E
A
H
G
C
B
Step 6
Have A
Execute
F&B –> Z
C&D –> F
A–> D
RULES


FACTS
E
A
H
G
C
B
Step 1
Z not here
Need to
Want Z
Get F
Get B
Z here
F&B –> Z
C&D –> F
A–> D
RULES
FACTS
E
A
H
G
C
B
Step 2
Need to
Want F
Get C
Get D
F here
F&B –> Z
C&D –> F
A–> D
RULES
FACTS
E
A
H
G
C
B
Step 3
C here
Want C
F&B –> Z
C&D –> F
A–> D
RULES

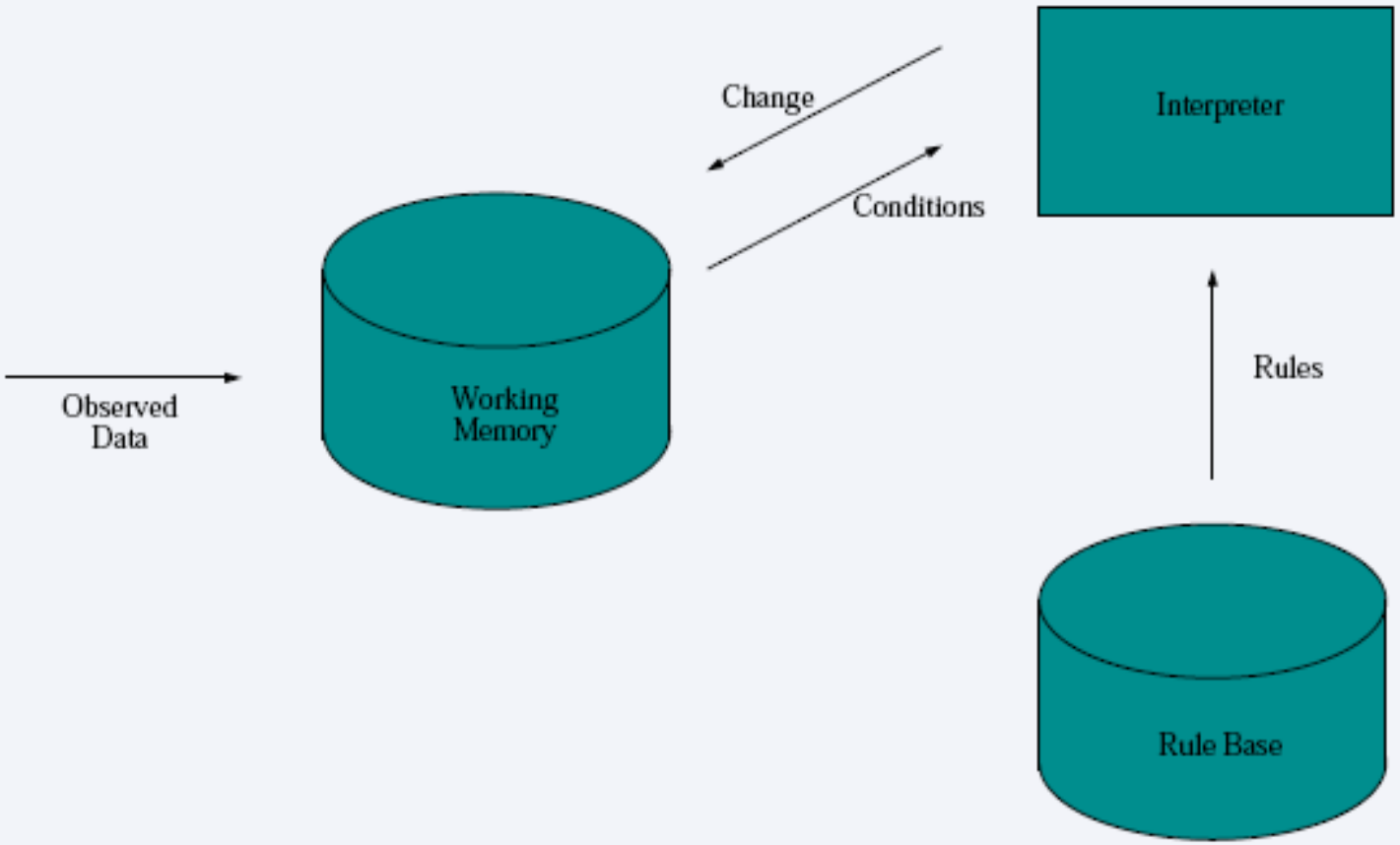

FACTS
E
A
H
G
C
B
D
Have C
Have D
Step 7
Execute
FACTS
E
A
H
G
C
F
B
D
Have F
Have B
Step 8
Execute
FACTS
E
A
H
G
C
F
B
D
Z
Have Z
F&B –> Z
C&D –> F
A–> D
RULES
F&B –> Z
C&D –> F
A–> D
RULES
F&B –> Z
C&D –> F
A–> D
RULES

Backward chaining algorithm:

‡ Prove goal G :

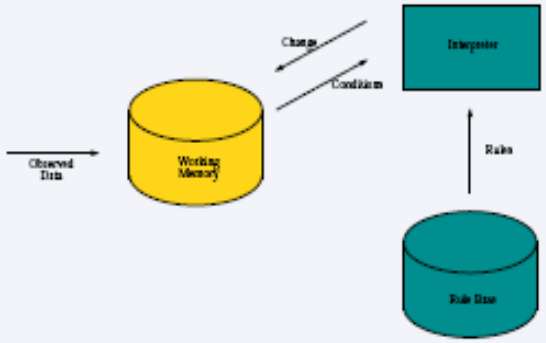
- If G is in the initial facts , it is proven.
- Otherwise, find a rule which can be used to conclude G, and try to prove each of that rule's conditions.

Component of Rule based System



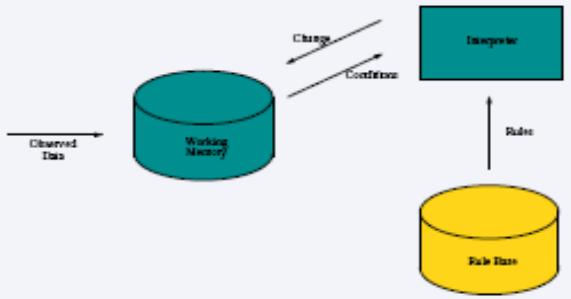
Working Memory:

- Contains facts about the world
  - Can be observed directly or derived from a rule
- Contains temporary knowledge – knowledge about this problem-solving session
- May be modified by the rules.
- Traditionally stored as a `<object, attribute, value>` triplet
- **Examples:**
  - `<CAR, COLOR, RED>`: "The color of my car is red"
  - `<TEMPERATURE, OVER, 20>`: "The temperature is over 20 C"



Rule base

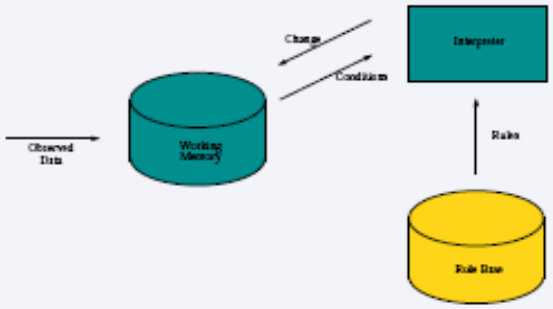
- Contains rules, each rule a step in a problem solving process.
- Rules are persistent knowledge about the domain.
- Typically only modified from the outside of the system, e.g. by an expert on the domain.



- The syntax is a `IF <conditions> THEN <actions>` format.
- **Examples:**
  - `IF <TEMPERATURE, OVER, 20> THEN add(<OCEAN, SWIMABLE, YES>)`
  - `IF <FEVER, OVER, 39> AND <NECK, STIFF, YES> AND <HEAD, PAIN, YES> THEN add(<PATIENT,DIAGNOSE, MENINGITIS>)`
- The conditions are matched to the working memory, and if they are fulfilled, the rule may be fired.

Rule base (cont…)

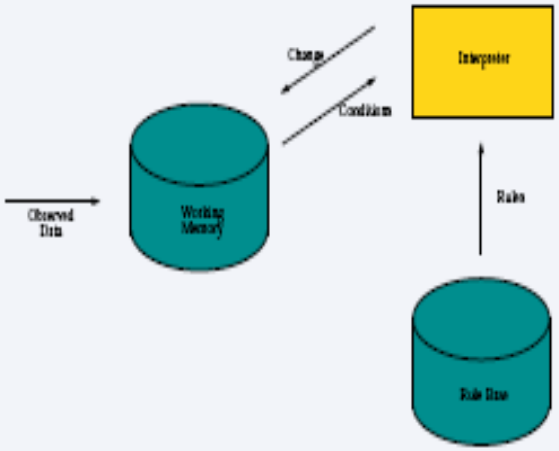
- Actions can be:
  - Adding fact(s) to the working memory.
  - Removing fact(s) from the working memory
  - Modifying fact(s) in the working memory.



- Systems can allow variables in the rules.
  - **Example:**
    - `IF <$x, ISA, CAR> AND <$x, LIGHTS, DIM> THEN add(<CHECK, BATTERY, $x>)`
  - Far more expressive rules → more computationally expensive inference.

Interpreter

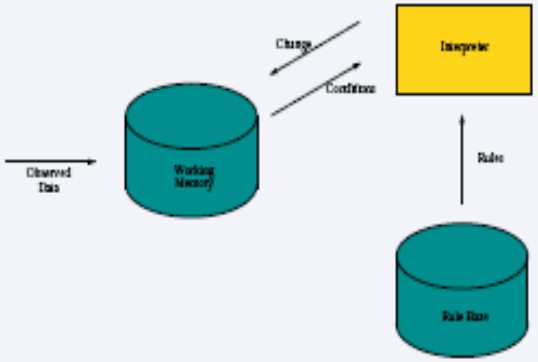
- Is the (domain independent) reasoning mechanism for Rule-Based Systems.
- Selects rule from the Rule Base to apply.



- The rules must match the current contents of the Working Memory.
- Applies the rule by performing the action.

Interpreter (cont…)

The Interpreter operates on a cycle:



Retrieval: Finds the rules that matches the current Working Memory. These rules are the Conflict Set.

Refinement: Prunes, reorders and resolves conflicts in the Conflict Set.

Execution: Executes the actions of the rules in the Conflict Set. Applies the rule by performing the action.