



مجموعة جامعة
الجميع الذكية
للتعليم القابضة



Master of Science in Artificial intelligent

Lecture 7

AI511

Foundations of Artificial Intelligence

Prepared by
Prof. Lamiaa Fattouh Ibrahim

www.midocean.ae

Outlines

- Recall: Rule-Base Expert System
- Forward chaining and backward chaining
- Example of Forward chaining and backward chaining
- Frames as a knowledge representation

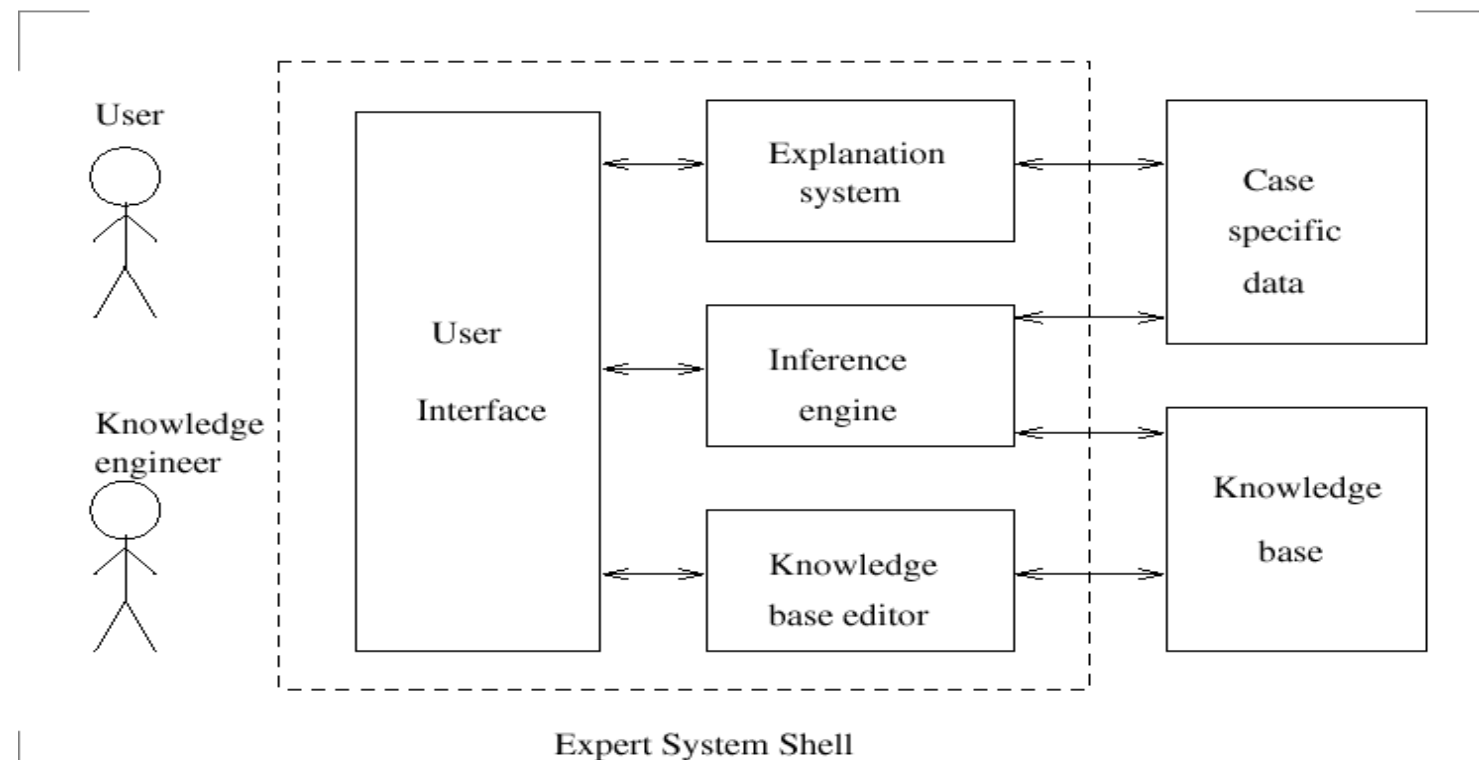
Outlines

Frame-based expert systems

- Introduction, or what is a frame?
- Frames as a knowledge representation technique
- Inference in frame-based expert systems
- Methods and demons
- Summary

Architecture of an Expert System I

Often there is separation between the domain specific knowledge (knowledge base and case specific data) and the rest of the system. The general purpose part is known as an *expert system shell*.



Guidelines to determine whether a problem is appropriate for expert system solution:

1. The need for the solution justifies the cost and effort of building an expert system.
2. Human expertise is not available in all situations where it is needed.
3. The problem may be solved using symbolic reasoning.
4. The problem domain is well structured and does not require commonsense reasoning.
5. The problem may not be solved using traditional computing methods.
6. Cooperative and articulate experts exist.
7. The problem is of proper size and scope.

Acquiring Knowledge

- Knowledge acquisition can be regarded as a method by which a **knowledge engineer** gathers information mainly from **experts**, but also from text books, technical manuals, research papers and other authoritative sources for ultimate translation into a **knowledge base**, understandable by both machines and humans.
- The person undertaking the knowledge acquisition, the knowledge engineer, must convert the acquired knowledge into an electronic format that a computer program can use.

Acquiring Knowledge

- The important characteristics of knowledge are that it is experiential, descriptive, qualitative, largely undocumented and constantly changing.
- There are certain domains where all these properties are found and some where there are only a few.
- The lack of documentation and the fact that experts carry a lot of information in their heads, makes it difficult to gain access to their knowledge for developing information systems in general and expert systems in particular.
- Therefore, knowledge engineers have devised specialised techniques to extract and document this information in an efficient and useful manner.

Production rules usually turn out to be a natural form of expressing knowledge, and have the following additional desirable features:

- **Modularity:** each rule define a small, relatively independent piece of knowledge.
- **Increment ability:** new rules can be added to the knowledge base relatively independently of other rules.
- **Modifiability (as a consequence of modularity):** old rules can be changed relatively independent of other rules.
- Support system's **transparency:** by transparency of the system we mean the system's ability to explain its decisions and solutions.

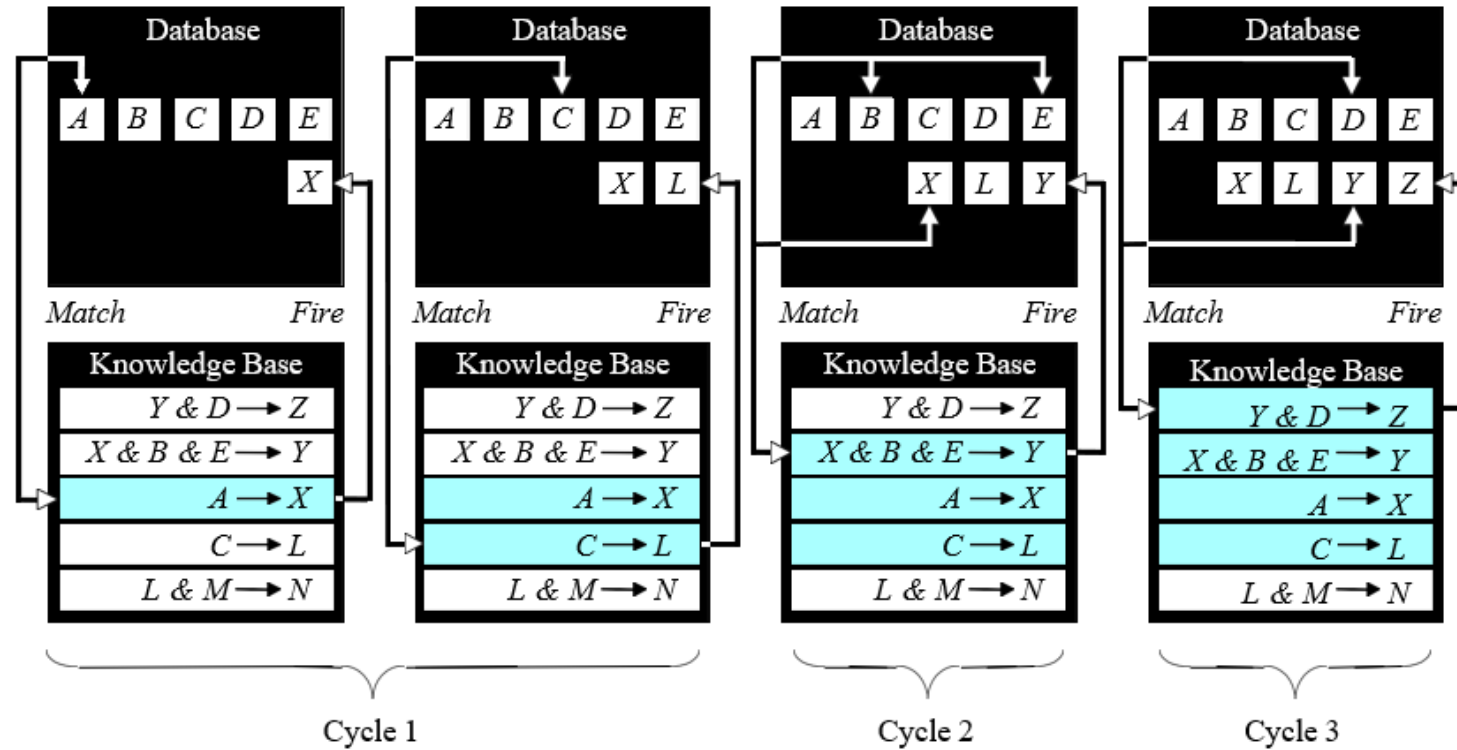
Production rules facilitate answering the following basic types of user's questions:

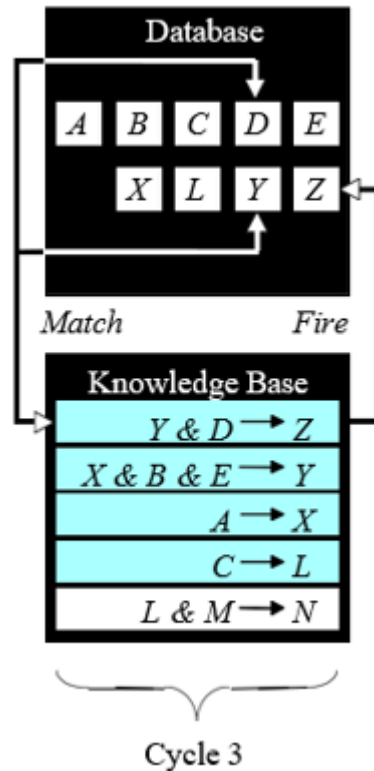
1. **"How"** questions: How did you reach this conclusion?
2. **"Why"** questions: Why are you interested in this information?

Forward chaining

- Forward chaining is the **data-driven reasoning**. The reasoning starts from the known data and proceeds forward with that data. Each time only the topmost rule is executed. When fired, the rule adds a new fact in the database. Any rule can be executed only once. The match-fire cycle stops when no further rules can be fired.

Forward chaining





- Forward chaining is a technique for gathering information and then inferring from it whatever can be inferred.
- However, in forward chaining, many rules may be executed that have nothing to do with the established goal.
- Therefore, if our goal is to infer only one particular fact, the forward chaining inference technique would not be efficient.

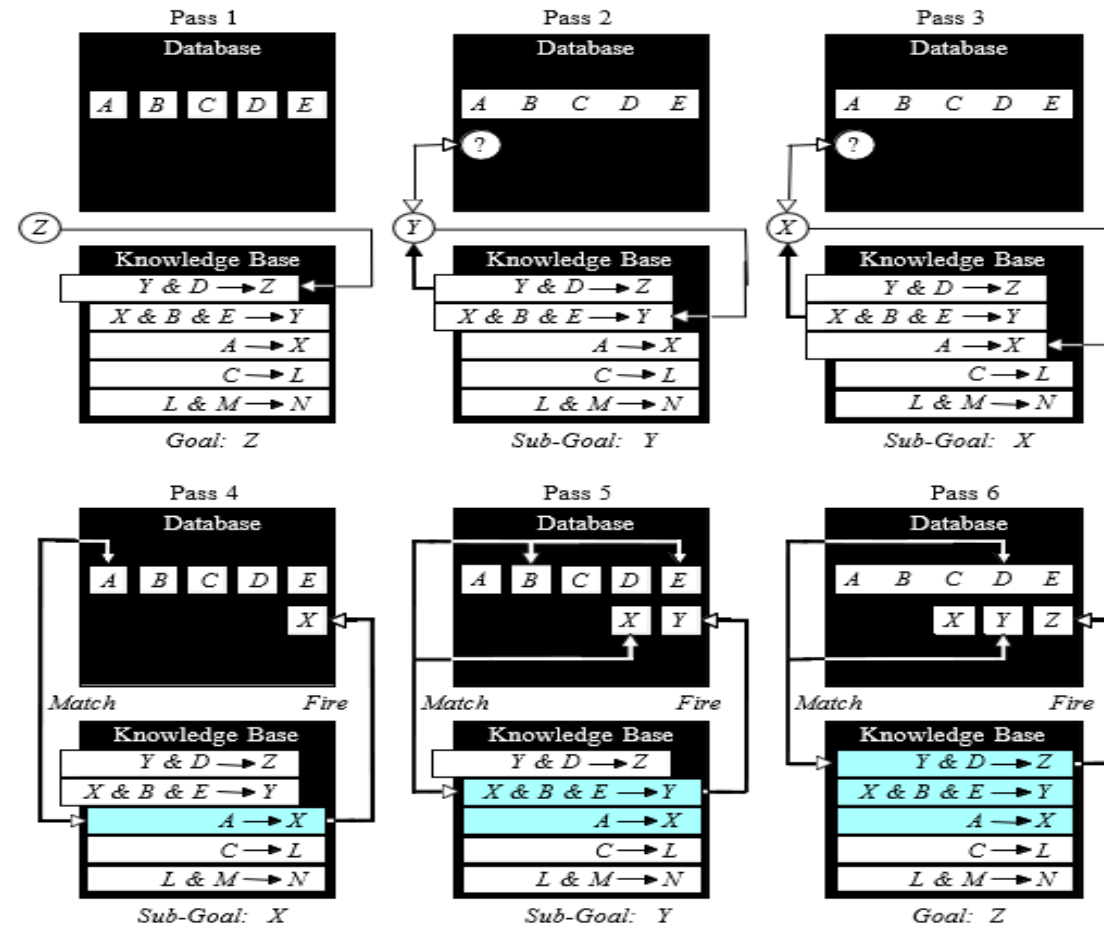
Backward chaining

- Backward chaining is the **goal-driven reasoning**. In backward chaining, an expert system has the goal (a **hypothetical solution**) and the inference engine attempts to find the evidence to prove it. First, the knowledge base is searched to find rules that might have the desired solution. Such rules must have the goal in their THEN (action) parts. If such a rule is found and its IF (condition) part matches data in the database, then the rule is fired and the goal is proved. However, this is rarely the case.

Backward chaining

- Thus the inference engine puts aside the rule it is working with (the rule is said to *stack*) and sets up a new goal, a subgoal, to prove the IF part of this rule. Then the knowledge base is searched again for rules that can prove the subgoal. The inference engine repeats the process of stacking the rules until no rules are found in the knowledge base to prove the current subgoal.

Backward chaining



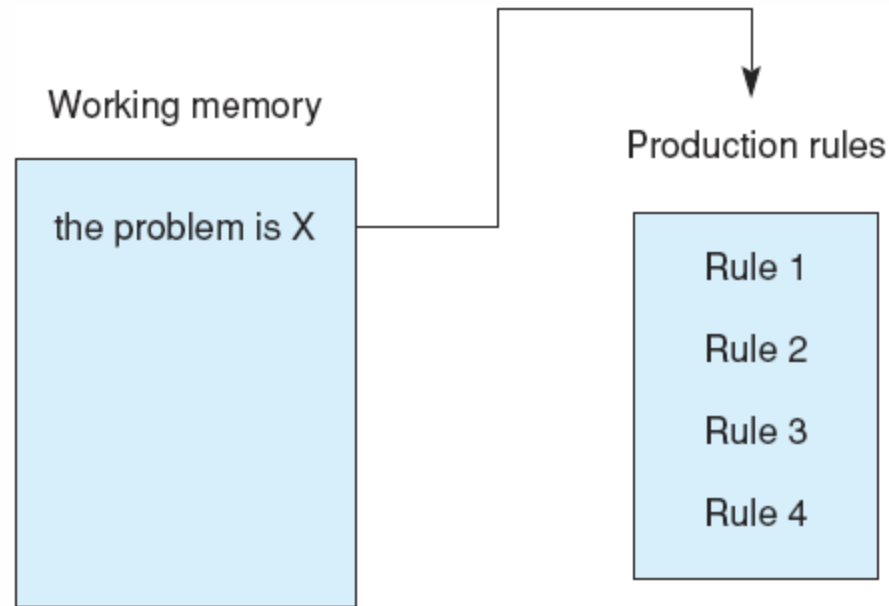
How do we choose between forward and backward chaining?

- If an expert first needs to gather some information and then tries to infer from it whatever can be inferred, choose the forward chaining inference engine.
- However, if your expert begins with a hypothetical solution and then attempts to find facts to prove it, choose the backward chaining inference engine.

A small expert system for analysis of automotive problems.

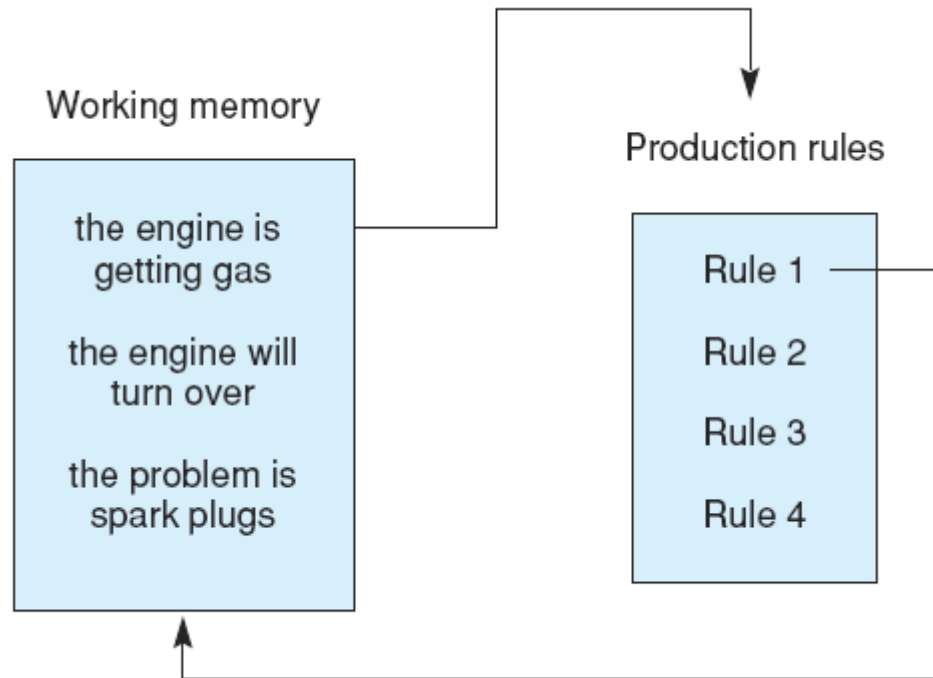
- Rule 1: if
 the engine is getting gas, and
 the engine will turn over,
 then
 the problem is spark plugs.
- Rule 2: if
 the engine does not turn over, and
 the lights do not come on
 then
 the problem is battery or cables.
- Rule 3: if
 the engine does not turn over, and
 the lights do come on
 then
 the problem is the starter motor.
- Rule 4: if
 there is gas in the fuel tank, and
 there is gas in the carburettor
 then
 the engine is getting gas.

Fig 8.5 The production system at the start of a consultation in the car diagnostic example (using goal-driven inference).



Place the top-level goal, the problem is X, in working memory as shown in figure. X is a variable that can match with any phrase, for example the problem is battery or cables; it will become bound to the solution when the problem is solved.

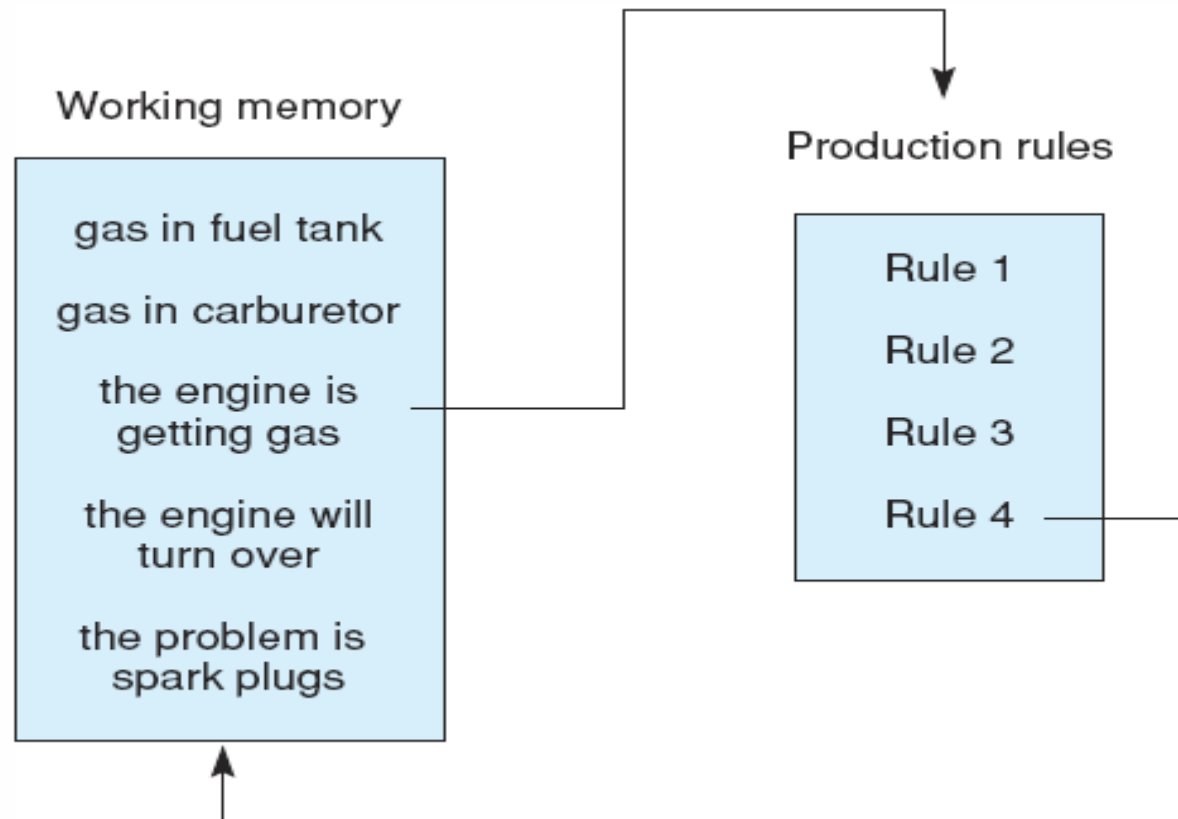
Fig 8.6 The production system after Rule 1 has fired.



- Rule 1: if
the engine is getting gas, and
the engine will turn over,
then
the problem is spark plugs.
- Rule 2: if
the engine does not turn over, and
the lights do not come on
then
the problem is battery or cables.
- Rule 3: if
the engine does not turn over, and
the lights do come on
then
the problem is the starter motor.
- Rule 4: if
there is gas in the fuel tank, and
there is gas in the carburettor
then
the engine is getting gas.

Fig 8.7 The system after Rule 4 has fired. Note the stack-based approach to goal reduction.

We may then fire rule 4, whose conclusion matches with the engine is getting gas, causing its premises to be placed in working memory.



- Rule 1: if
the engine is getting gas, and
the engine will turn over,
then
the problem is spark plugs.
- Rule 2: if
the engine does not turn over, and
the lights do not come on
then
the problem is battery or cables.
- Rule 3: if
the engine does not turn over, and
the lights do come on
then
the problem is the starter motor.
- Rule 4: if
there is gas in the fuel tank, and
there is gas in the carburettor
then
the engine is getting gas.

The following dialogue begins with the computer asking the user about the goals present in working memory.

Gas in fuel tank?

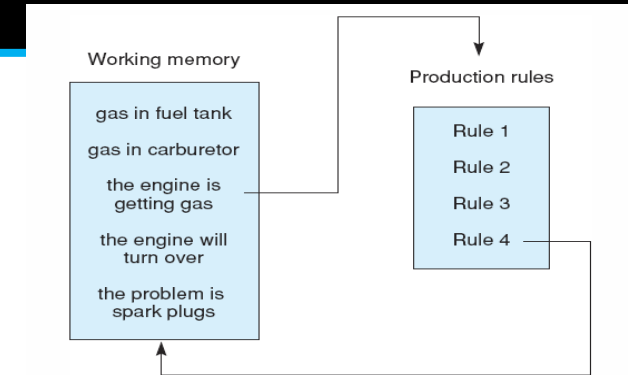
Yes

Gas in carburettor?

Yes

Engine will turn over?

Why



here the user enters a why query, i.e., why do you want to know whether the engine will turn over? The system responds with a literal presentation of the rule under question:

It has been established that:

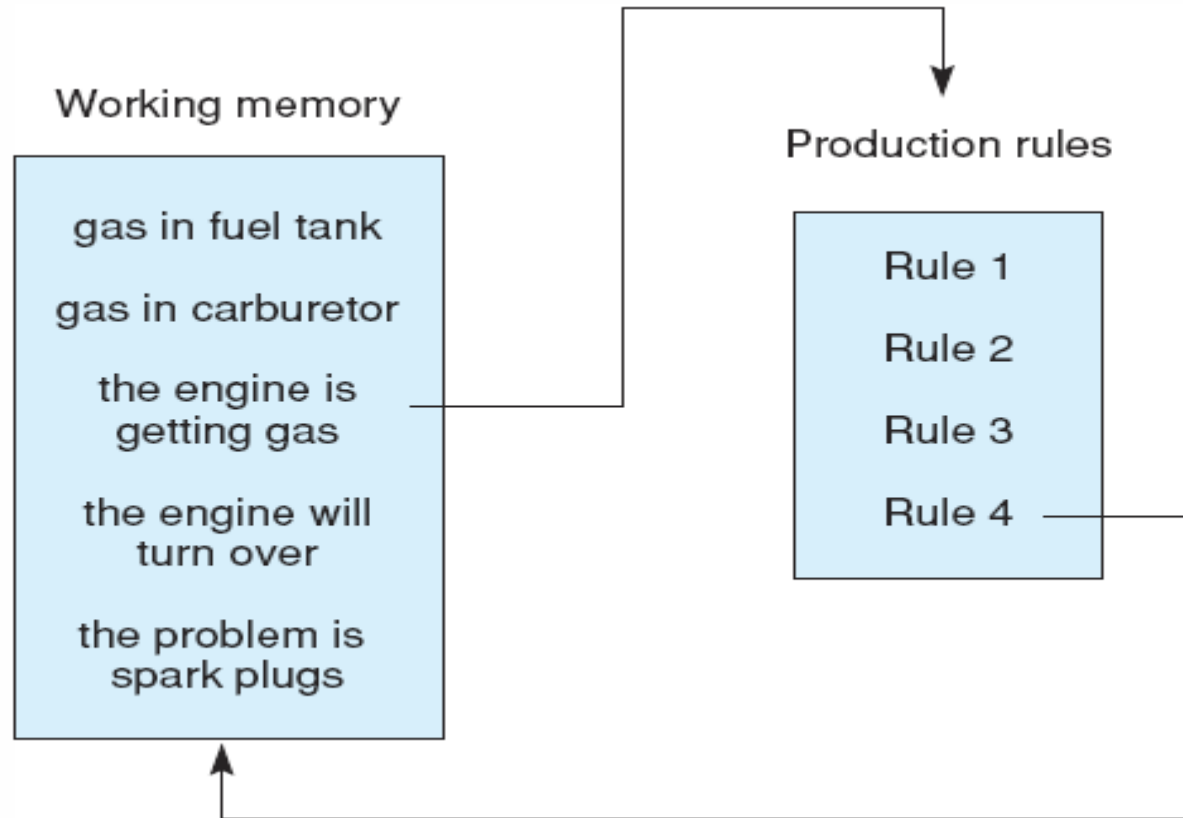
1. The engine is getting gas,

Therefore if

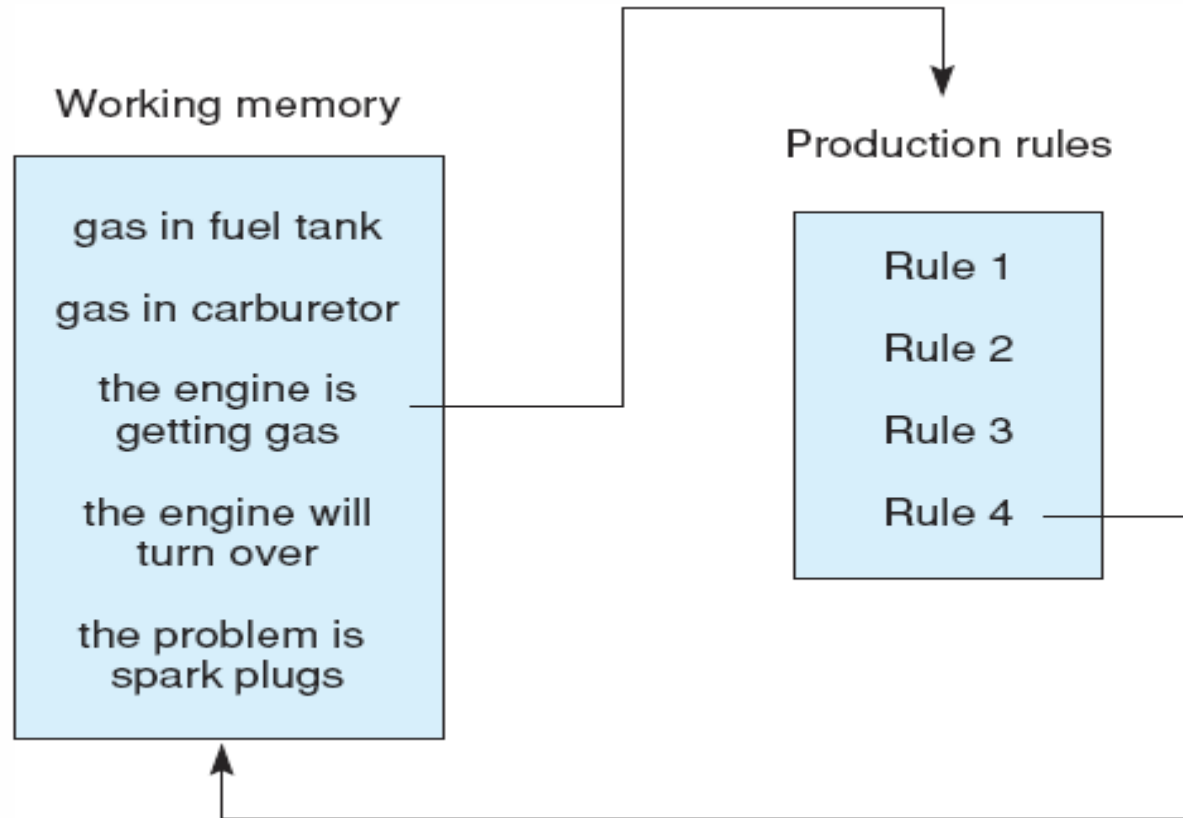
2. The engine will turn over,

Then the problem is the spark plugs.

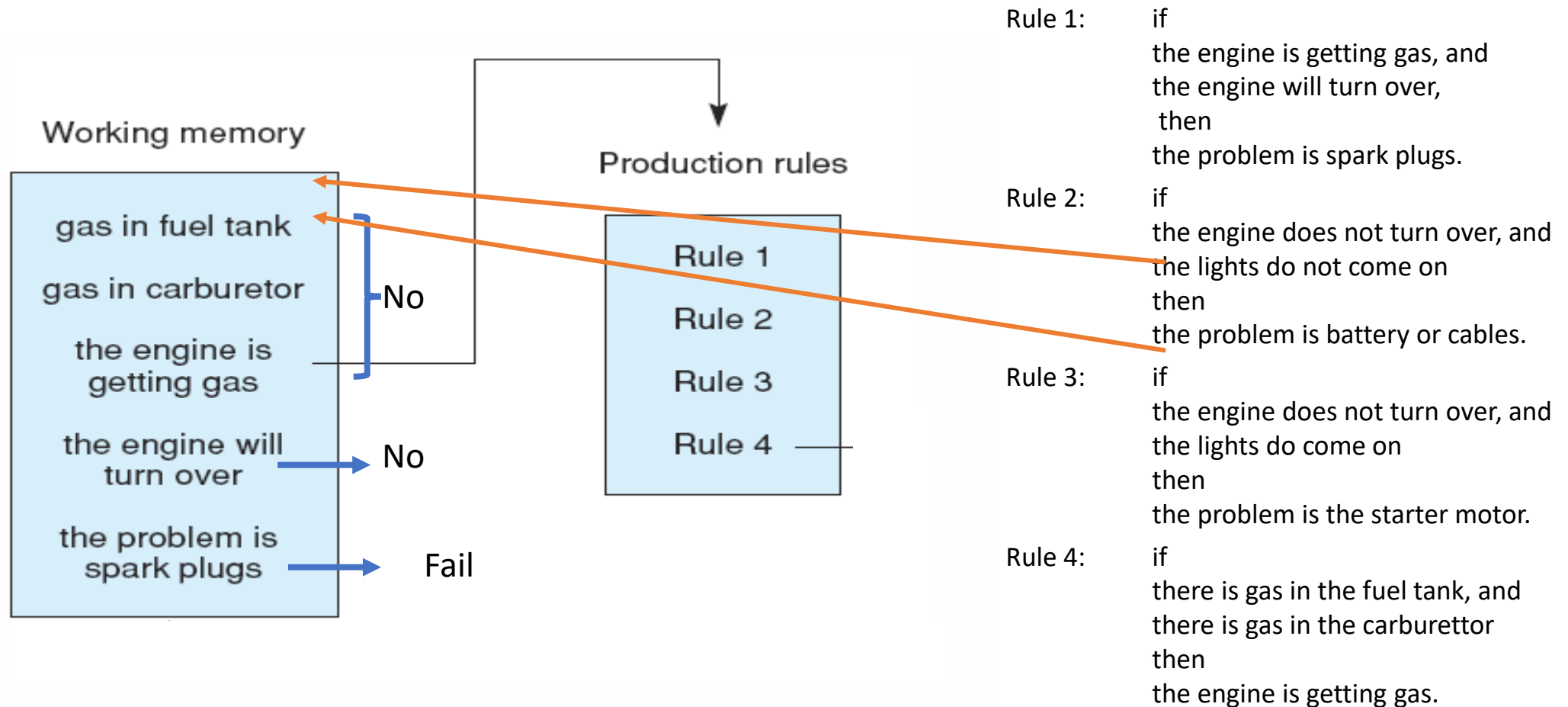
Because the user answered yes to the first two questions, rule 4 was able to fire, verifying the first premise of rule 1. When the why query was made, the system was trying to solve the second premise of rule 1. Note that the explanation for the system's behaviour is just a restatement of the current rule under consideration.



- Rule 1: if
the engine is getting gas, and
the engine will turn over,
then
the problem is spark plugs.
- Rule 2: if
the engine does not turn over, and
the lights do not come on
then
the problem is battery or cables.
- Rule 3: if
the engine does not turn over, and
the lights do come on
then
the problem is the starter motor.
- Rule 4: if
there is gas in the fuel tank, and
there is gas in the carburetor
then
the engine is getting gas.



- Rule 1: if
the engine is getting gas, and
the engine will turn over,
then
the problem is spark plugs.
- Rule 2: if
the engine does not turn over, and
the lights do not come on
then
the problem is battery or cables.
- Rule 3: if
the engine does not turn over, and
the lights do come on
then
the problem is the starter motor.
- Rule 4: if
there is gas in the fuel tank, and
there is gas in the carburettor
then
the engine is getting gas.



Forward chaining

Using the Production System for Data-Driven Reasoning

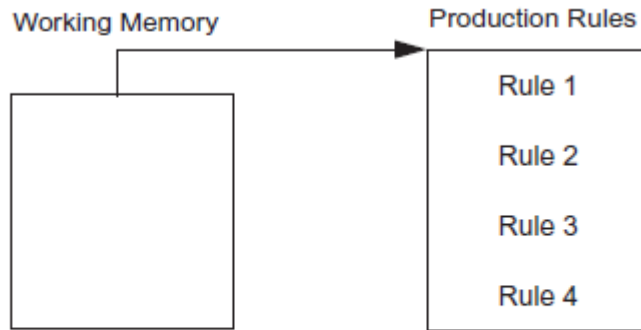


Figure 8.9 The production system at the start of a consultation for data-driven reasoning.

Breadth-first search is even more common in data-driven reasoning. The algorithm for this is simple: compare the contents of working memory with the conditions of each rule in the rule base according to the order of the rules in the rule base. If the data in working memory supports a rule's firing the result is placed in working memory and then control moves on to the next rule. Once all rules have been considered, search starts again at the beginning of the rule set.

- Rule 1: if
the engine is getting gas, and
the engine will turn over,
then
the problem is spark plugs.
- Rule 2: if
the engine does not turn over, and
the lights do not come on
then
the problem is battery or cables.
- Rule 3: if
the engine does not turn over, and
the lights do come on
then
the problem is the starter motor.
- Rule 4: if
there is gas in the fuel tank, and
there is gas in the carburettor
then
the engine is getting gas.

If a piece of information that makes up (part of) the premise of a rule is not the conclusion of some other rule then that fact will be deemed “askable” when control comes to the situation (rule) where that information is needed. For example, the engine is getting gas is not askable in the premise of rule 1, because that fact is a conclusion of another rule, namely rule 4.

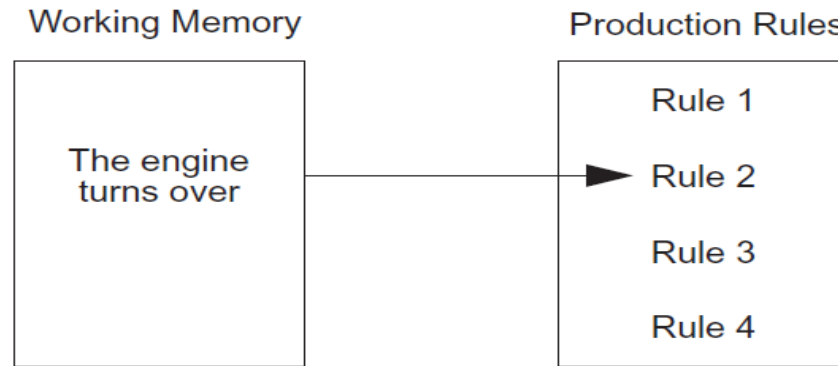


Figure 8.10 The production system after evaluating the first premise of Rule 2, which then fails.

We first examine premises of the four rules in order to see what information is “askable”. The premise, **the engine is getting gas**, is **not askable**, so rule 1 fails and control moves to rule 2.

The engine does not turn over is askable. Suppose the answer to this query is **false**, so **the engine will turn over** is placed in working memory

- Rule 1: if
the engine is getting gas, and
the engine will turn over,
then
the problem is spark plugs.
- Rule 2: if
the engine does not turn over, and
the lights do not come on
then
the problem is battery or cables.
- Rule 3: if
the engine does not turn over, and
the lights do come on
then
the problem is the starter motor.
- Rule 4: if
there is gas in the fuel tank, and
there is gas in the carburettor
then
the engine is getting gas.

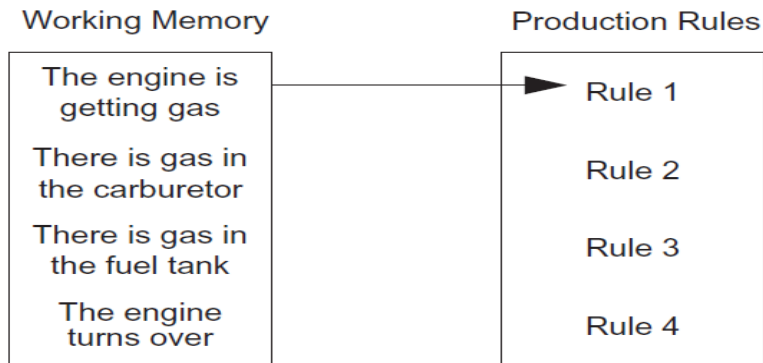


Figure 8.11 The data-driven production system after considering Rule 4, beginning its second pass through the rules.

But rule 2 fails, since the first of two **and** premises is false, and consideration moves to rule 3, where again, the first premise fails. At rule 4, both premises are askable. Suppose the answer to both questions is true. Then there is gas in the fuel tank and there is gas in the carburetor are placed in working memory, as is the conclusion of the rule, the engine is getting gas.

- Rule 1: if
the engine is getting gas, and
the engine will turn over,
then
the problem is spark plugs.
- Rule 2: if
the engine does not turn over, and
the lights do not come on
then
the problem is battery or cables.
- Rule 3: if
the engine does not turn over, and
the lights do come on
then
the problem is the starter motor.
- Rule 4: if
there is gas in the fuel tank, and
there is gas in the carburettor
then
the engine is getting gas.

At this point all the rules have been considered so search now returns, with the new contents of working memory, to consider the rules in order a second time. When the working memory is matched to rule 1, its conclusion, the problem is spark plugs, is placed in working memory. In this example no more rules will match and fire, and the problem-solving session is completed

- Rule 1: if
the engine is getting gas, and
the engine will turn over,
then
the problem is spark plugs.
- Rule 2: if
the engine does not turn over, and
the lights do not come on
then
the problem is battery or cables.
- Rule 3: if
the engine does not turn over, and
the lights do come on
then
the problem is the starter motor.
- Rule 4: if
there is gas in the fuel tank, and
there is gas in the carburettor
then
the engine is getting gas.

Introduction, or what is a frame?

A frame is a data structure with typical knowledge about a particular object or concept.

Frames were first proposed by Marvin Minsky in the 1970s.

Boarding pass frames

QANTAS BOARDING PASS

Carrier: QANTAS AIRWAYS
Name: MR N BLACK
Flight: QF 612
Date: 29DEC
Seat: 23A
From: HOBART
To: MELBOURNE
Boarding: 0620
Gate: 2

AIR NEW ZEALAND BOARDING PASS

Carrier: AIR NEW ZEALAND
Name: MRS J WHITE
Flight: NZ 0198
Date: 23NOV
Seat: 27K
From: MELBOURNE
To: CHRISTCHURCH
Boarding: 1815
Gate: 4

- Each frame has its own name and a set of attributes, or slots, associated with it. *Name, weight, height* and *age* are slots in the frame *Person*.
- *Model, processor, memory* and *price* are slots in the frame *Computer*. Each attribute or slot has a value attached to it.
- Frames provide a natural way for the structured and concise representation of knowledge.
- A frame provides a means of organising knowledge in slots to describe various attributes and characteristics of the object.
- Frames are an application of object-oriented programming for expert systems.

- Object-oriented programming is a programming method that uses *objects* as a basis for analysis, design and implementation.
- In object-oriented programming, an object is defined as a concept, abstraction or thing with crisp boundaries and meaning for the problem at hand. All objects have identity and are clearly distinguishable. *Michael Black, Audi 5000 Turbo, HP Pavilion P6555A* are examples of objects.

- An object combines both data structure and its behaviour in a single entity.
- This is in sharp contrast to conventional programming, in which data structure and the program behaviour have concealed or vague connections.
- When an object is created in an object-oriented programming language, we first assign a name to the object, then determine a set of attributes to describe the object's characteristics, and at last write procedures to specify the object's behaviour.
- A knowledge engineer refers to an object as a frame (the term, which has become the AI jargon).

Frames as a knowledge representation technique

- The concept of a frame is defined by a collection of slots. Each slot describes a particular attribute or operation of the frame.
- Slots are used to store values. A slot may contain a default value or a pointer to another frame, a set of rules or procedure by which the slot value is obtained.

Typical information included in a slot

- Frame name.
- Relationship of the frame to the other frames.
The frame *HP Pavilion P6555A* might be a member of the class *Computer*, which in turn might belong to the class *Hardware*.
- Slot value. A slot value can be symbolic, numeric or Boolean. For example, the slot *Name* has symbolic values, and the slot *Age* numeric values. Slot values can be assigned when the frame is created or during a session with the expert system.

- Default slot value. The default value is taken to be true when no evidence to the contrary has been found. For example, a car frame might have four wheels and a chair frame four legs as default values in the corresponding slots.
- Range of the slot value. The range of the slot value determines whether a particular object complies with the stereotype requirements defined by the frame. For example, the cost of a computer might be specified between \$750 and \$1500.
- Procedural information. A slot can have a procedure attached to it, which is executed if the slot value is changed or needed.

- Frame-based expert systems also provide an extension to the slot-value structure through the application of *facets*.
- A facet is a means of providing extended knowledge about an attribute of a frame.
- Facets are used to establish the attribute value, control end-user queries, and tell the inference engine how to process the attribute.

What are the class and instances?

- The word *frame* often has a vague meaning. The frame may refer to a particular object, for example the computer *HP Pavilion P6555A*, or to a group of similar objects. To be more precise, we will use the *instance-frame* when referring to a particular object, and the *class-frame* when referring to a group of similar objects.
- A class-frame describes a group of objects with common attributes.
- *Animal*, *person*, *car* and *computer* are all class-frames.
- Each frame “knows” its class.

Computer class

CLASS:	<i>Desktop</i>	
[Str] <i>Item Code:</i>		
[Str] <i>Brand:</i>		
[Str] <i>Model:</i>		
[Str] <i>Processor:</i>		
[Str] <i>Installed RAM:</i>		
[Str] <i>Hard Drive:</i>		
[Str] <i>Graphical Card:</i>		
[Str] <i>Optical Drive:</i>	[Default]	← DVD Burner
[Str] <i>USB Ports:</i>		
[Str] <i>Keyboard:</i>	[Default]	← 104-key
[Str] <i>Warranty:</i>	[Default]	← 1 year
[N] <i>Cost:</i>		
[Str] <i>Stock:</i>	[Initial]	← In stock

Computer instances

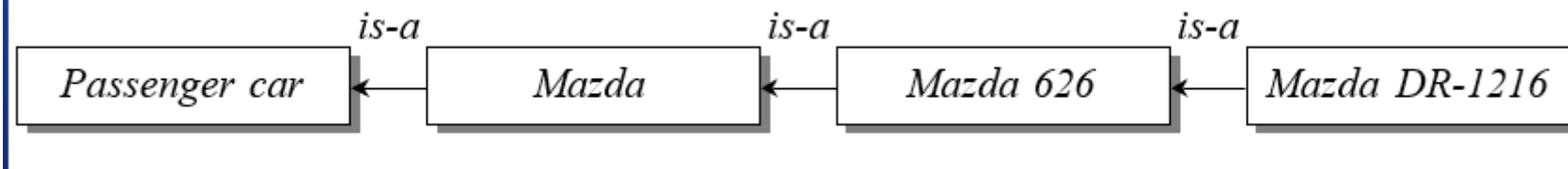
INSTANCE: <i>HP Pavilion P655A</i>	
<i>Class:</i>	<i>Desktop</i>
[Str] <i>Item Code:</i>	<i>SY7973</i>
[Str] <i>Brand:</i>	<i>HP</i>
[Str] <i>Model:</i>	<i>Pavilion P655A</i>
[Str] <i>Processor:</i>	<i>Intel Core i5 3.2 GHz</i>
[Str] <i>Installed RAM:</i>	<i>4 GB</i>
[Str] <i>Hard Drive:</i>	<i>1 TB</i>
[Str] <i>Graphical Card:</i>	<i>1 GB</i>
[Str] <i>Optical Drive:</i>	<i>DVD Burner</i>
[Str] <i>USB Ports:</i>	<i>8</i>
[Str] <i>Keyboard:</i>	<i>104-key</i>
[Str] <i>Warranty:</i>	<i>1 year</i>
[N] <i>Cost:</i>	<i>1296.00</i>
[Str] <i>Stock:</i>	<i>In stock</i>

INSTANCE: <i>Acer Aspire M3910</i>	
<i>Class:</i>	<i>Desktop</i>
[Str] <i>Item Code:</i>	<i>SY7975</i>
[Str] <i>Brand:</i>	<i>Acer</i>
[Str] <i>Model:</i>	<i>Aspire M3910</i>
[Str] <i>Processor:</i>	<i>Intel Core i7 2.8 GHz</i>
[Str] <i>Installed RAM:</i>	<i>8 GB</i>
[Str] <i>Hard Drive:</i>	<i>2 TB</i>
[Str] <i>Graphical Card:</i>	<i>1 GB</i>
[Str] <i>Optical Drive:</i>	<i>DVD Burner</i>
[Str] <i>USB Ports:</i>	<i>10</i>
[Str] <i>Keyboard:</i>	<i>104-key</i>
[Str] <i>Warranty:</i>	<i>1 year</i>
[N] <i>Cost:</i>	<i>2147.00</i>
[Str] <i>Stock:</i>	<i>In stock</i>

Class inheritance in frame-based systems

- Frame-based systems support class inheritance.
- The fundamental idea of inheritance is that attributes of the class-frame represent things that are *typically* true for all objects in the class.
- However, slots in the instance-frames can be filled with actual data uniquely specified for each instance.

Relations of the car frames



Inheritance of slot values

CLASS:	<i>Passenger car</i>
[C]	<i>Engine type</i> <i>In-line 4</i> <i>cylinder: V6:</i>
[N]	<i>Horsepower:</i>
[C]	<i>Drivetrain type</i> <i>Rear wheel</i> <i>drive: Front</i> <i>wheel drive:</i> <i>Four wheel</i> <i>drive:</i>
[C]	<i>Transmission type</i> <i>5-speed manual:</i> <i>4-speed automatic:</i>
[N]	<i>Fuel consumption (mpg):</i>
[N]	<i>Seating capacity:</i>

CLASS:	<i>Mazda</i>
Superclass:	<i>Passenger car</i>
[C]	<i>Engine type</i> <i>In-line 4</i> <i>cylinder: V6:</i>
[N]	<i>Horsepower:</i>
[C]	<i>Drivetrain type</i> <i>Rear wheel</i> <i>drive: Front</i> <i>wheel drive:</i> <i>Four wheel</i> <i>drive:</i>
[C]	<i>Transmission type</i> <i>5-speed manual:</i> <i>4-speed automatic:</i>
[N]	<i>Fuel consumption (mpg):</i>
[N]	<i>Seating capacity:</i>
[Str]	<i>Country of manufacture:</i> <i>Japan</i>

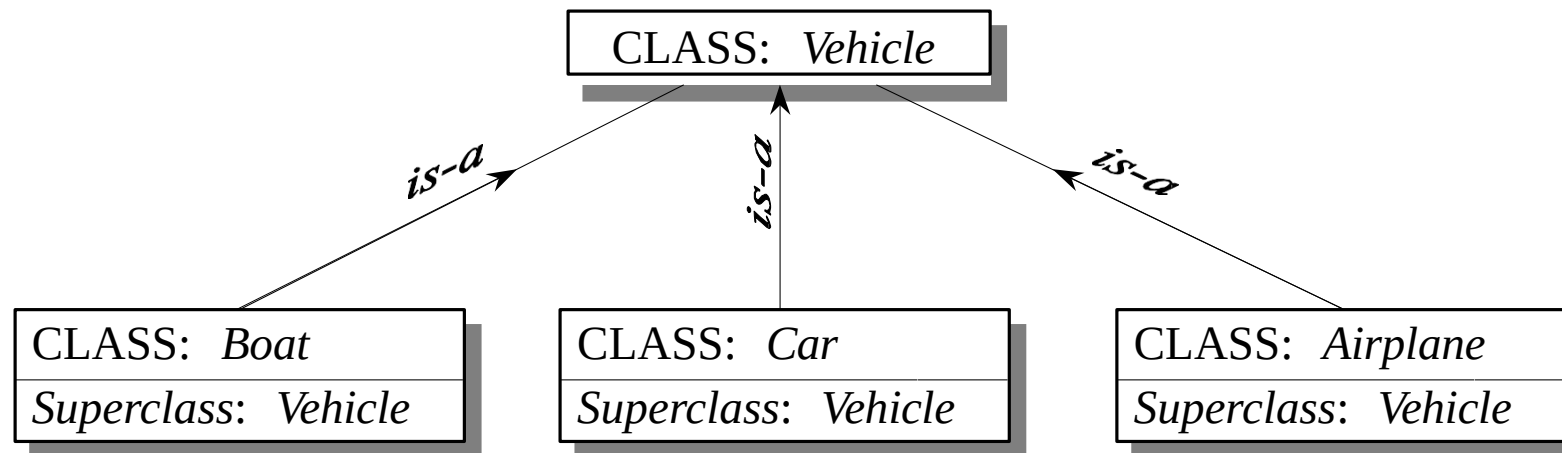
Inheritance of slot values (continued)

CLASS:	<i>Mazda 626</i>	
Superclass:	<i>Mazda</i>	
[C]	<i>Engine type</i>	
	<i>In-line 4</i>	
	<i>cylinder: V6:</i>	
[N]	<i>Horsepower:</i>	<i>125</i>
[C]	<i>Drivetrain type</i>	
	<i>Rear wheel</i>	
	<i>drive: Front</i>	
	<i>wheel drive:</i>	
	<i>Four wheel</i>	
	<i>drive:</i>	
[C]	<i>Transmission type</i>	
	<i>5-speed manual:</i>	
	<i>4-speed automatic:</i>	
[N]	<i>Fuel consumption (mpg):</i>	<i>22</i>
[N]	<i>Seating capacity:</i>	<i>5</i>
[Str]	<i>Country of manufacture:</i>	<i>Japan</i>
[Str]	<i>Model:</i>	
[C]	<i>Colour</i>	
	<i>Glacier White:</i>	
	<i>Sage Green</i>	
	<i>Metallic: Slate Blue</i>	
	<i>Metallic: Black</i>	
	<i>Onyx Clearcoat:</i>	
[Str]	<i>Owner:</i>	

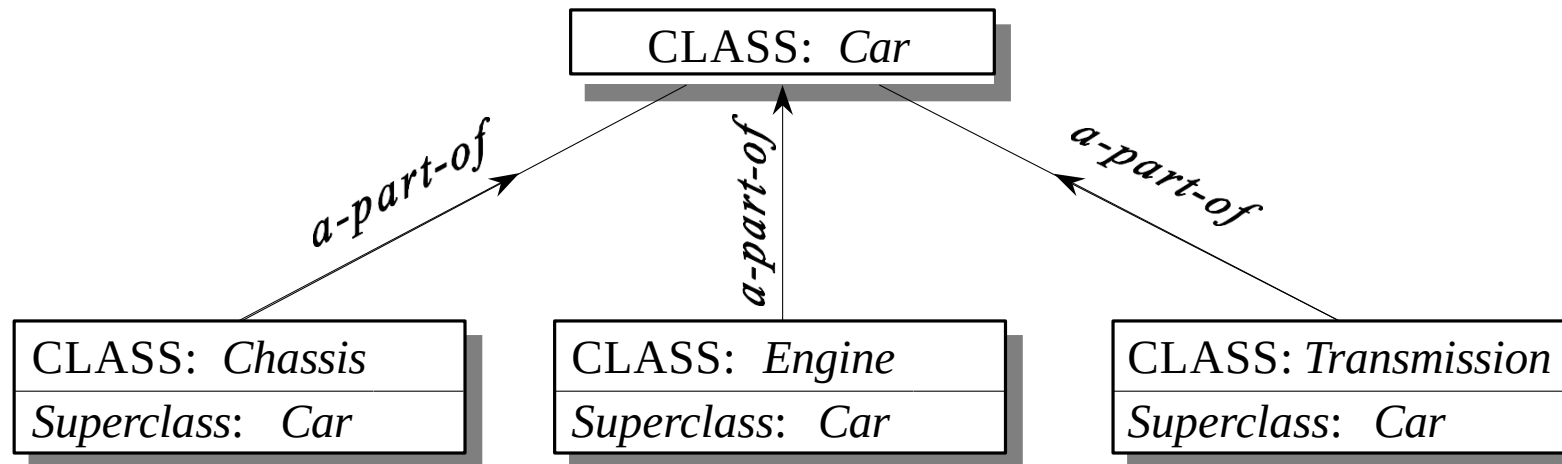
INSTANCE:	<i>Mazda DR-1216</i>	
Class:	<i>Mazda 626</i>	
[C]	<i>Engine type</i>	
	<i>In-line 4</i>	TRUE
	<i>cylinder: V6:</i>	FALSE
[N]	<i>Horsepower:</i>	<i>125</i>
[C]	<i>Drivetrain type</i>	
	<i>Rear wheel drive:</i>	FALSE
	<i>Front wheel drive:</i>	TRUE
	<i>Four wheel drive:</i>	FALSE
[C]	<i>Transmission type</i>	
	<i>5-speed manual:</i>	FALSE
	<i>4-speed automatic:</i>	TRUE
[N]	<i>Fuel consumption (mpg):</i>	<i>28</i>
[N]	<i>Seating capacity:</i>	<i>5</i>
[Str]	<i>Country of manufacture:</i>	<i>Japan</i>
[Str]	<i>Model:</i>	<i>DX</i>
[C]	<i>Colour</i>	
	<i>Glacier White:</i>	FALSE
	<i>Sage Green Metallic:</i>	TRUE
	<i>Slate Blue Metallic:</i>	FALSE
	<i>Black Onyx Clearcoat:</i>	FALSE
[Str]	<i>Owner:</i>	<i>Mr Black</i>

Relationships among objects

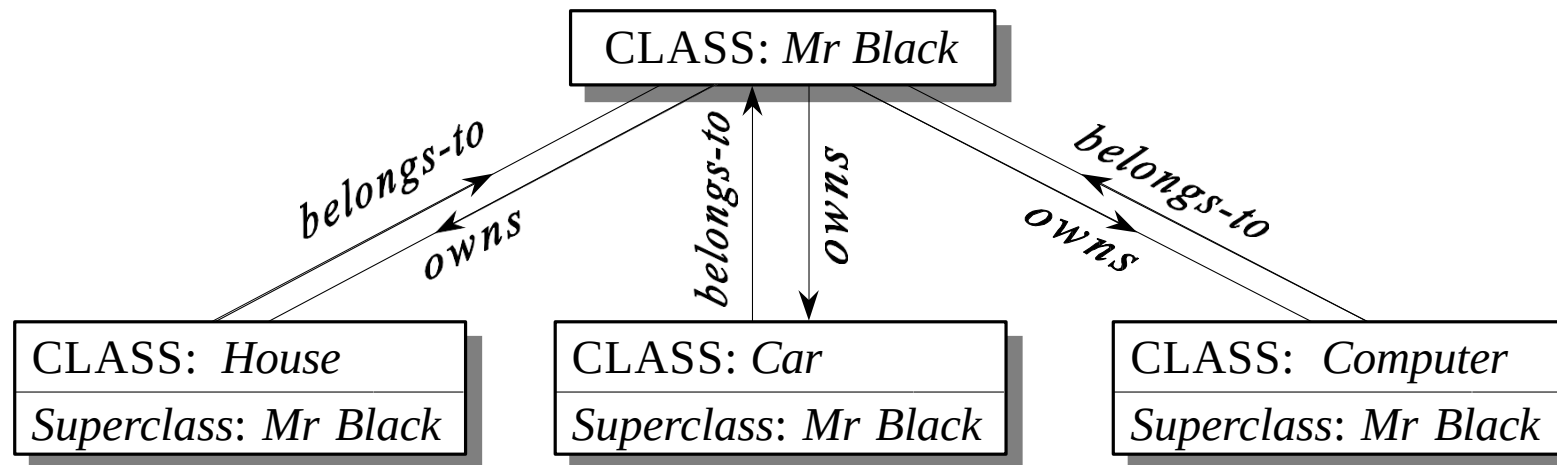
- Generalisation denotes *a-kind-of* or *is-a* relationship between superclass and its subclasses. For example, a car *is a* vehicle, or in other words, *Car* represents a subclass of the more general superclass *Vehicle*. Each subclass inherits all features of the superclass.



- Aggregation is *a-part-of* or *part-whole* relationship in which several subclasses representing *components* are associated with a superclass representing a *whole*. For example, an engine is *a part of* a car.



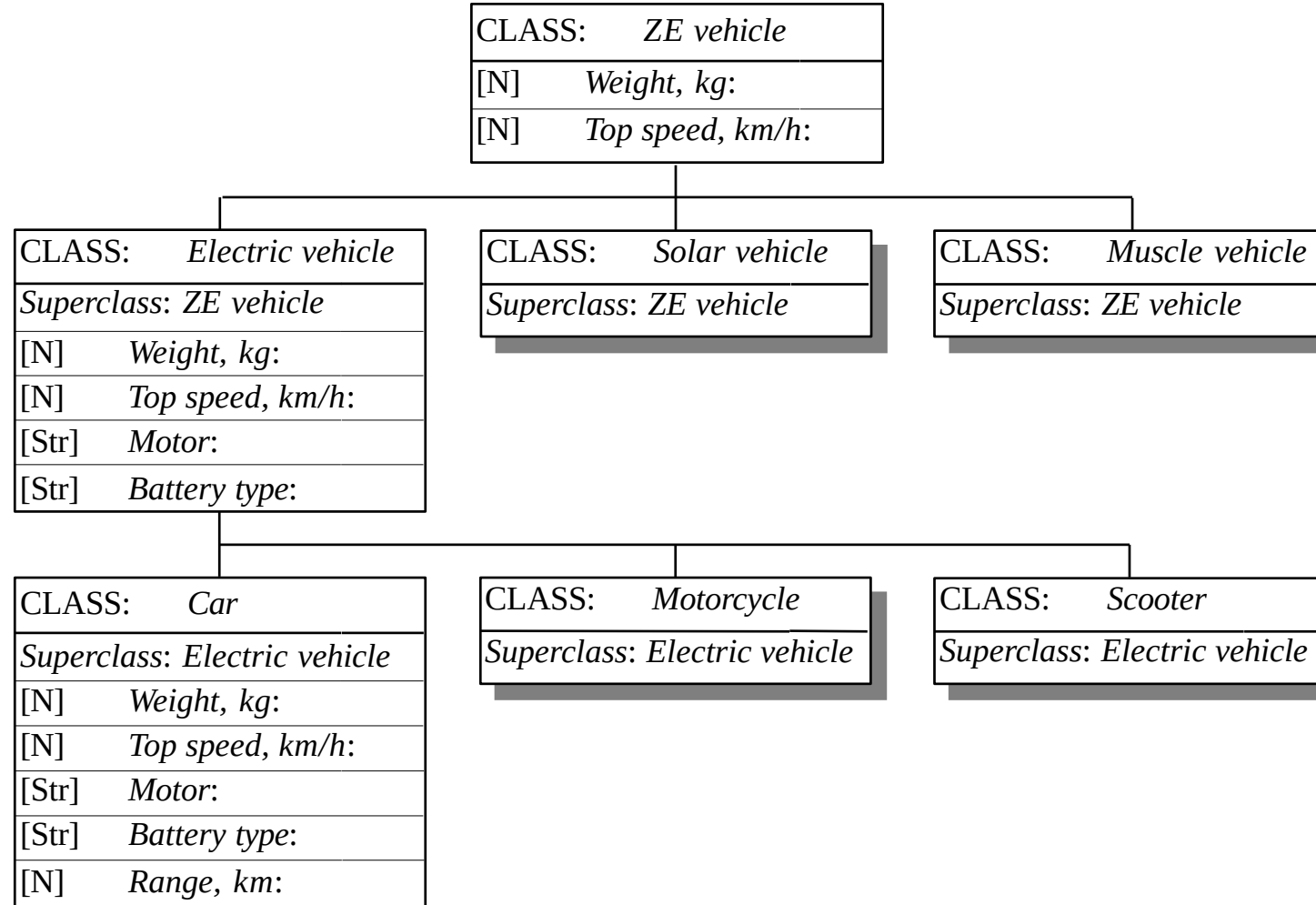
- Association describes some semantic relationship between different classes which are unrelated otherwise. For example, Mr Black owns a house, a car and a computer. Such classes as *House*, *Car* and *Computer* are mutually independent, but they are linked with the frame *Mr Black* through the semantic association.



Inheritance in frame-based systems

- Inheritance is defined as the process by which all characteristics of a class-frame are assumed by the instance-frame.
- A common use of inheritance is to impose default features on all instance-frames. We can create just one class-frame that contains generic characteristics of some object, and then obtain several instance-frames without encoding the class-level characteristics

One-parent inheritance in zero-emission vehicles

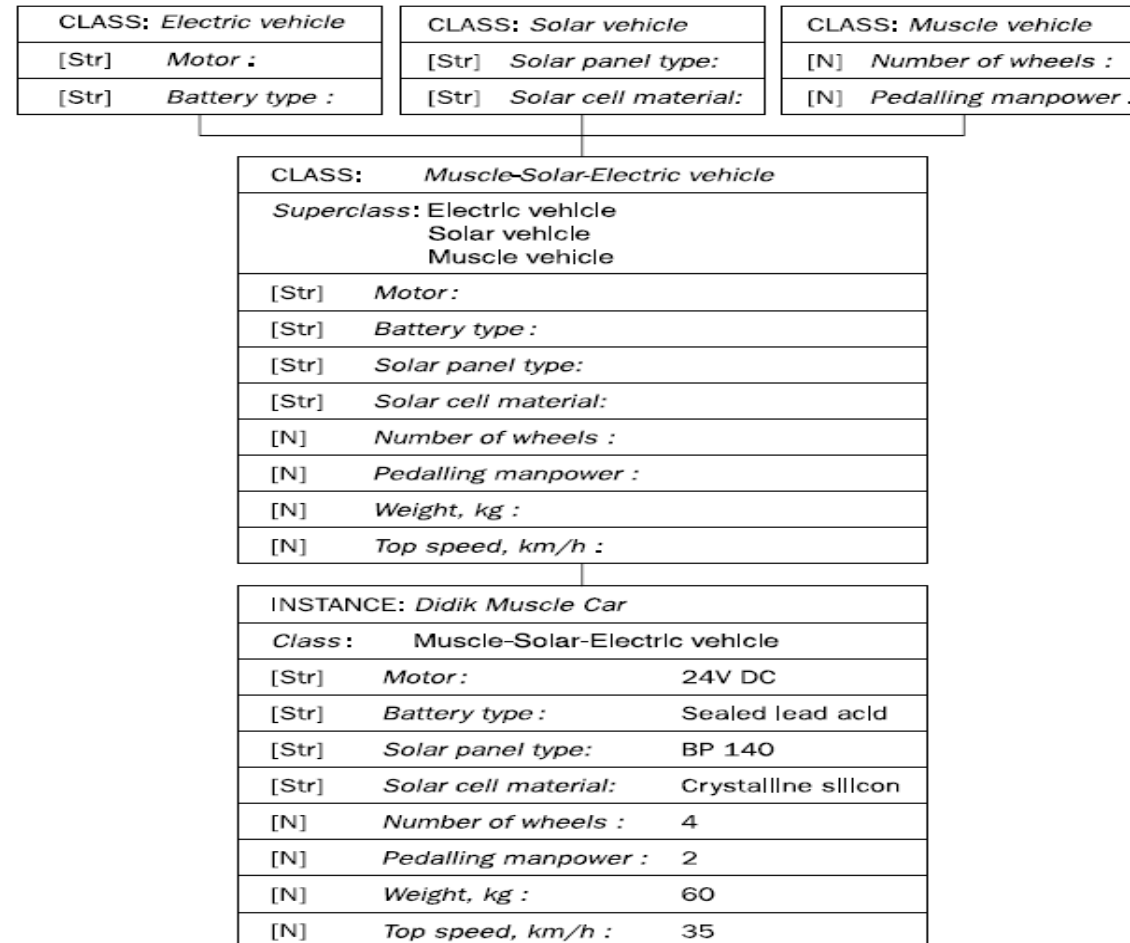


One-parent inheritance (continued)

CLASS:	Van
Superclass:	Car
[N]	Weight, kg:
[N]	Top speed, km/h:
[Str]	Motor:
[Str]	Battery type:
[N]	Range, km:
[N]	Payload, kg:

INSTANCE:	<i>Ford Ecostar</i>	
Class:	<i>Van</i>	
[N]	<i>Weight, kg:</i>	<i>1851</i>
[N]	<i>Top speed, km/h:</i>	<i>113</i>
[Str]	<i>Motor:</i>	<i>3-phase, AC induction</i>
[Str]	<i>Battery type:</i>	<i>Sodium sulfur</i>
[N]	<i>Range, km:</i>	<i>161</i>
[N]	<i>Payload, kg:</i>	<i>463</i>

One-parent inheritance (continued)



- Expert systems are required not only to store the knowledge but also to validate and manipulate this knowledge. To add actions to our frames, we need methods and demons.

- A method is a procedure associated with a frame attribute that is executed whenever requested.
- We write a method for a specific attribute to determine the attribute's value or execute a series of actions when the attribute's value changes.
- Most frame-based expert systems use two types of methods:
 - **WHEN CHANGED** and **WHEN NEEDED**.

What is the difference between methods and demons?

- A demon has an IF-THEN structure. It is executed whenever an attribute in the demon's IF statement changes its value. In this sense, demons and methods are very similar, and the two terms are often used as synonyms.
- However, methods are more appropriate if we need to write complex procedures. Demons, on the other hand, are usually limited to IF-THEN statements.

Summary

- Rule-Base Expert System
- Forward chaining and backward chaining
- Example of Forward chaining and backward chaining
- Frames as a knowledge representation



مجموعة جامعة
الجميع الذكية
للتعليم القابضة

MIDOCEAN
UNIVERSITY
BY EVERYONE'S SMART UNIVERSITY



Thank You

