Lecture one:

Contents:

1. ***introduction to prolog language***
2. ***some o f prolog language characteristic***
3. ***prolog language uses***
4. ***prolog language component***

4.1 fact

4.2 rule

4.3 questions

5. Variables

1. Introduction to prolog language

Prolog: is a computer programming language that is used for solving problems involves objects and relationships between objects.

Example:

"John owns the book"

Owns (john,book) relationship(object1,object2)

The relationship has a specific order, johns own the book, but the book dose not owns john, and this relationship and its representation above called fact.

♦we are using rule to describe relationship between objects.

Example: the rule" two people are sisters if they are both female and have the same parents"

1. Tell us something about what it means to be sisters.
2. Tell us hoe to find if two people are sisters, simply: check to see if they are both female have the same parents.

Component of computer programming in prolog

Computer programming in prolog consist of:

1. Declearing some facts about object and their relationships.
2. Defining some rules about objects and their relationships.
3. Asking questions about objects and their relationships.

if we write our rule about sisters, we could then ask the questions whether Mary and Jane are sisters.

Prolog would search through what we told it about Mary and Jane, and come back with the answer Yes or No, depending on what we told it earlier.

So, we can consider prolog as a store house of facts and rules, and it uses the facts and rules to answer questions.

♦prolog is a conversational language. Which means you and the computer carry out a kind of conversation, typing a letter from keyboard and displaying it at the screen, prolog work like this manner, prolog will wait for you to type in facts and rules that certain to the problem you want to solve? Then if you ask the right kind of questions prolog will work out the answers and show them.

2.Some of prolog language characteristics:

1. We can solve a particular problem using prolog in less no of line of code.
2. It's an important tool to develop AI application and ES.
3. Prolog program consist of fact and rule to solve the problem and the output is all possible answer to the problem.
4. Prolog language is a descriptive language use the inference depend on fact and rule we submit to get all possible answer while in other language the programmer must tell the computer on how to reach the solution by gives the instruction step by step.

3. Prolog language uses:

1. Construct NLI (Natural Language Interface).
2. Translate language.
3. Constructor symbolic manipulation language packages.
4. Implement powerfully database application.
5. Construct expert system programs.

4. Prolog language component

4.1 Facts

Is the mechanism for representing knowledge in the program.

Syntax of fact:

1. The name of all relationship and objects must begin with a lower-case letter, for example likes (john, mary).

2. The relationship is written first, and the objects are written separated by commas, and enclosed by a pair of round brackets.

Like (john, mary)

3. The full stop character '.' Must come at the end of fact.

Example:

Gold is valuable valuable (gold).

Jane is female female (jane).

John owns gold owns (johns, gold).

Johns is the father of Mary father (john, marry).

The names of objects that are enclosed within the round brackets are called arguments. And the name of relationship called predicates Relationship has arbitrary number of argument. If we want to define predicate called play, were we mention two players and a game they play with each other, it can be:

Play (john, Mary, football).

In prolog the collection of facts is called database.

4.2 Rules

Rules are used when you want to say that a fact depends on a group of other facts, and we use the following syntax:

1. One fact represents the head (conclusion).
2. The word if used after the head and represented as ":-".
3. One or more fact represents the requirement (condition).

The syntax of if statement

If (condition) then (conclusion)

[Conclusion: - condition] rule

Example:

I use the umbrella if there is rain

Conclusion condition

Represent both as fact like:

Wheatear (rain).

Use (umbrella)

Use (Iam, umberella):-whether (rain).

4.3 Questions

Question used to ask about facts and rules.

Question look like the fact and written under the goal program section while fact and rule written under clauses section.

Example: for the following fact owns (mary , book).

We can ask:dose mary own the book in the following manner:

Goal:

Owns (mary ,book)

When Q is asked in prolog, it will search through the database you typed before, it look for facts that match the fact in the question.

Two fact matches if their predicates are the same and their corresponding argument are the same, if prolog finds a fact that matches the question, prolog will respond with Yes, otherwise the answer is No.

5. Variables

If we want to get more interest information about fact or rule, we can use variable to get more than Yes/No answer.

*variables dose not name a particular object but stand for object that we cannot name.

*variable name must begin with capital letter.

*using variable we can get all possible answer about a particular fact or rule.

*variable can be either bound or not bound.

Variable is bound when there is an object that the variable stands for.

The variable is not bound when what the variable stand for is not yet known.

Example:

Fact

*L*ike (john, mary).

Like (john, flower).

Like (ali, mary).

Question:

1. Like (john,X)

X= mary

X = flower

2. like(X, mary)

X=john

3. Like(X, Y)

X=john Y=flower

X=john Y=mary

X=ali Y=mary

5. Type of questing in the goal

There are three type of question in the goal summarized as follow:

1. Asking with constant: prolog matching and return Yes/No answer.

2. Asking with constant and variable: prolog matching and produce

result for the Variable.

3. Asking with variable: prolog produce result.

Example:

Age(a,10).

Age(b,20).

Age(c,30).

Goal:

1.Age(a,X). ans:X=10 Type2

2.age(X,20). Ans:X=b Type2

3.age(X,Y). ans: X=a Y=10, X=b Y=20, X=c Y=30. Type3

4.Age(_,X). ans:X=10 , X=20, X=30. '_' means don't care Type3

5.Age(_,_). Ans:Yes Type1

H.W:

Convert the following paragraph into fact or rule:

1. a person may steal something if the person is a thief and he likes the

thing and the thing is valuable.

2. Bob likes all kind of game. Football is a game. Anything anyone plays

and not killed by is a game.

Lecture two:

Propositional logic

Ali is a brave man

This car has 4 wheels

Symbols	operator
P	not T
Q	V or
	^ and
	≡

equivalent

If weather is cold then it is winter

P Q

P →Q

Condition evident or conclusion

Laws:

~(~P) ≡ *P*

P →Q ≡ ~*P*V*Q*

*P*V*Q* ≡QV P

P^ Q ≡ Q^P

~ (PV Q) ≡ ~P^ ~Q

~ (P^ Q) ≡ ~P V~Q

Predicate Logic:

Relation معناها الفعل او الصفة

Object الاشياء التي نوصفها اما فعلا او صفة

Ali is a man

Man(ali)

is(ali,man)

object(obj1,obj2,…..).

1-Facts

Maha is a girl

Girl(maha)

Is(maha,girl).

I have a book

Have (I) book

Ali is a brave man

Is (ali , man, brave)

Man (ali,brave)

Brave (ali,man).

Man(ali) ^ brave (ali)

Ali have red car

Have (ali, car,red)

Have (ali,car) ^ colour(car,red)

This is sunny day

Is(day,sunny)

Sunny(day).

Maha has 4 books

Have(maha,4,book)

Have (maha,book) ^number (book,4)

Ali going to school now

go(ali,school) ^time(now)

I have one or two books

Have (I,books) ^ (number(books,1) v number(books,two))

2- rules

If its winter then it is cold

Is(weather,winter) →is (weather,cold)

When I'm sick , I will go to the doctor

Sick(I'm) → go (I,doctor)

I f student will read good he will pass

Read(X,good) → pass(X).

Ahmed got to the school when he is 6 years old

Age (ahmed,6)→ go(ahmed,school).

Example:

Write predicate for book in library

Book("artificial intelligence",'gorge lugur",2009,10)

Book("c++","xt",2009,9).

Example: write predicate for cars

Car("BMW","black","1990","full automatic", "special").

Car("Mazda","white","1995","ordinarily", "special").

Car("chery","yello","2009","full automatic", "Taxi").

Calling types:-

Book(“prolog”,”A.I”,”Gourge”,10,2000).

Book(“c++”,”programming”,”Rintice Hill”,5,2001).

Book(“Expert system”,”A.I”,”Daniel”,5,1994).

Goal: book(X,Y,A,b,C).

false predicate عدم المطابقة بسبب اختلاف اسم

No

Goal: book(A,B,C,D)

No arguments عدم مطابقة بسبب اختلاف عدد

Goal:book(A,B,C,D,E)

A=Prolod, B=A.I , C=George , D=10, E=2000

A=c++, B=programming, C=printce hill, D= 5, E=2001

A=expert system , B=A.I, C= Daniel, D=5, E=1994.

3/ solution

Yes

Goal: book(A,”A.I”),C,D,X).

A=prolog, C=George ,D=10,X=2000

A=expert, C=Daniel , D=5,X=2000.

2/SOLUTIONS

Goal: book(X,Y,Z,10,W).

X=Expert, Y=A.I , Z=George, W=2000

1/ SOLUTION

Goal: book(“prolog”,A,C,N,2000)

A=A.I , C=George, N=10

1/ solution

Goal: book(A,”A.I”,X,Y,2004).

No Solution

Goal: book(c++,A,B,20,X).

No Solution

ملاحظة مهمة جدا: المتغيرات Variables بلغة برولوك يكتب ب Capital letter.

Goal: Book(“c++”,”programming”,”Rintice Hill”,5,2001).

Yes

Book(“C++”,”A.I”,”Gourge”,10,2000).

NO

ملاحظة مهمة جدا : اذا اخذ ثابت قيمة معينة لايجوز تغييرها في نفس ال Predicate.

H.W

Goal: BOOK(A,”A.I”,N,P,2000).

Goal: book(A,”A.I”,X,5,1993).

Family Relations

Son(ali,ahmed).

Son(ahmed,majed).

Son(mohammed,taha).

Son(Hamza,ahmed).

Son(hussain,majed).

Son (Hassan ,hussain).

Father (X,Y):-son(X,Y).

Brother(X,Y):-father(Z,X),dather(Z,Y).

Grandfather(X,Y):-father(X,Z),father(Z,Y).

Cousin(X,Y):-father(Z,X),father(W,Y),brother(Z,W).

Goal

Father(,ali,B).

B=ahmed. Yes

Goal

Brother(hamza,C)

↓↠ Father(Z,hamza) , father(Z,C)

↓↠ Son(hamza,Z) true son(C,ahmed) true

Z=ahmed C=ali

H.W

Write appropriate predicates for the following family relations:

- Uncle
- Mother
- Sister

Lecture three:

Conjunctions and backtracking

1. Conjunctions
    1. and ‘,’.
    2. or ‘;’.

Used to combine facts in the rule , or to combine fact in the goal to answer questions about more complicated relationship.

Example:

Facts

Like (mary,food).

Like(mary,wine).

Like(john,mary).

Goal

Like(mary,john),like(john,mary).

We can ask dose mary like john and dose john like mary?

Now, how would prolog answer this complicated question?

Prolog answers the question by attempting to satisfy the first the first goal. if the first goal is in the database, then prolog will mark the place in the database, and attempt to satisfy the second goal.

If the second goal is satisfied, then prolog marks that goal‘s place in the database, and we have a solution that satisfy both goals.

♦ It is important to remember that each goal keeps its own place marker. If, however, the second goals are not satisfied, then prolog will attempt to re-satisfy the previous goal.

Prolog searches the database in case it has to re-satisfy the goal at a later time. But when a goal needs to be re-satisfied, prolog will begin the search the search database completely for each goal. If a fact in the database happens to match , satisfying the goal, then prolog will mark the place in the database in case it has to re-satisfy the goal at the later time. But when a goal needs to be re-satisfied, prolog will begin the search from the goal's own place marker, rather than from the start of database and this behavior called "backtracking".

Example: about backtracking

*Facts

Like(mary,food).

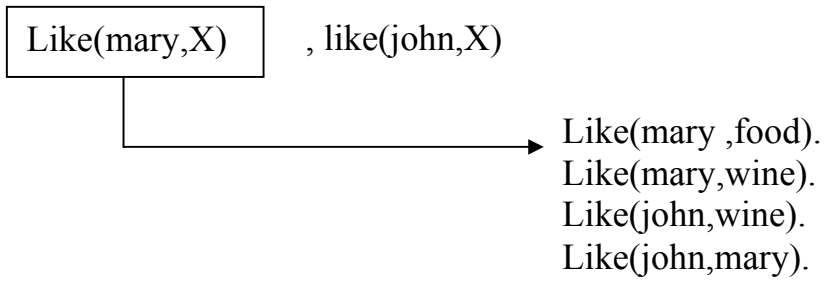
Like(mary,wine).

Like(john,wine).

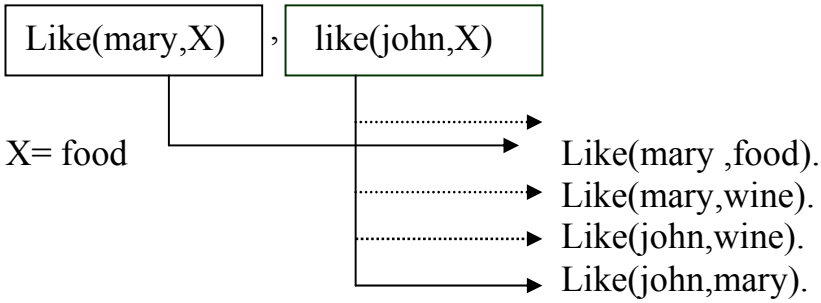
Like(john,mary).

*Goal:

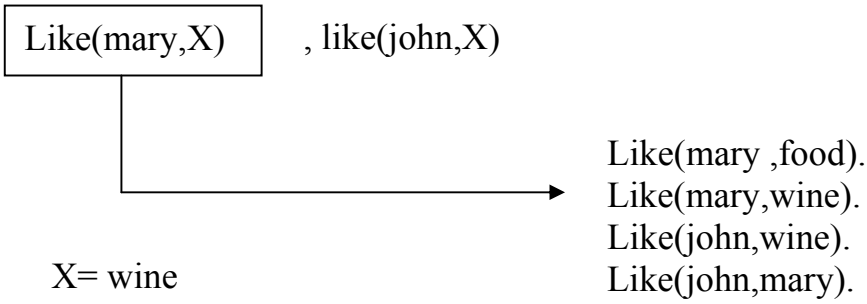
Like(mary,X),like(john,X).



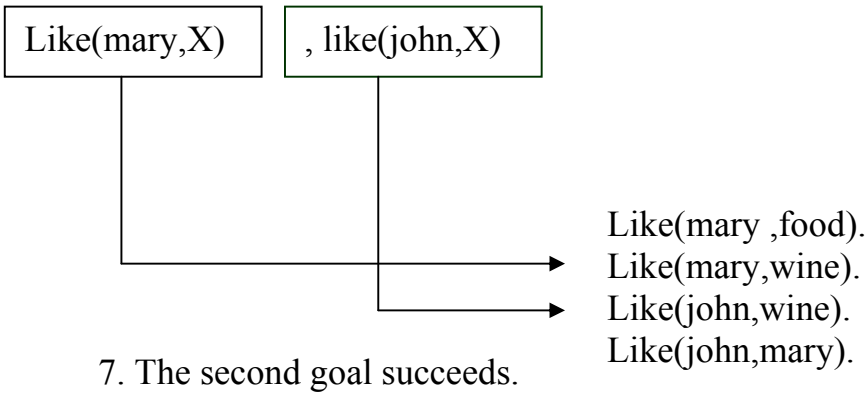
1. The first goal succeed, bound X to food.
2. Next, attempt to satisfy the second goal.



3. The second goal fails.
4. Next, backtrack: forget previous value of X and attempt to resatisfy the first goal.



5. The first goal succed agin, bund X to wine.
6. Next, attempt to satisfy the second goal.



7. The second goal succeeds.
8. Prolog notifies you of success.

H.W
Trace the following goal to find the value of X,Y,W,Z.
Fact
Mark(a,10).
Mark(b,20).

Mark(c,30).

Goal:
Mark(X,Y),Mark(W,Z).

Lecture four:

Content

1. Data type.

2. Program structure.

3. Read and write functions.

4. Arithmetic and logical operation.

1. data type

Prolog supports the following data type to define program entries.

1. **Integer:** to define numerical value like 1, 20, 0,-3,-50, ect.
2. **Real:** to define the decimal value like 2.4, 3.0, 5,-2.67, ect.
3. **Char:** to define single character, the character can be of type small letter or capital letter or even of type integer under one condition it must be surrounded by single quota. For example, 'a','C','123'.
4. **string :** to define a sequence of character like "good" i.e define word or statement entries the string must be surrounded by double quota for example "computer", "134", "a". The string can be of any length and type.
5. **Symbol:** anther type of data type to define single character or sequence of character but it must begin with small letter and don't surround with single quota or double quota.

2. program structure

Prolog program structure consists of five segments, not all of them must appear in each program. The following segment must be included in each program predicates, clauses, and goal.

1. **Domains:** define global parameter used in the program.

Domains
I= integer
C= char
S = string
R = real

2. **Data base:** define internal data base generated by the program
Database
Greater (integer)

3. **Predicates:** define rule and fact used in the program.
Predicates
Mark(symbol,integer).

4. **Clauses:** define the body of the program.. For the above predicates the clauses portion may contain Mark (a, 20).

5**.Goal:** can be internal or external, internal goal written after clauses portion , external goal supported by the prolog compiler if the program syntax is correct
This portion contains the rule that drive the program execution.

2. mathematical and logical operation

a .mathematical operation:

operation	symbol
addition	+
subtraction	-
multiplication	*
Integer part of division	div
Remainder of division	mod

B .logical operation

operation	symbol
greater	>
Less than	<
Equal	=
Not equal	<>
Greater or equal	>=
Less than or equal	<=

3. Other mathematical function

Function name	operation
Cos(X)	**Return the cosine of its argument**
Sine(X)	**Return the sine of its argument**
Tan(X)	**Return the tranget of its argument**
Exp(X)	**Return e raised to the value to which X is bound**
Ln(X)	**Return the natural logarithm of X (base e)**
Log(X)	**Return the base 10 logarithm of log 10^x**
Sqrt(X)	**Return the positive square of X**
Round(X)	**Return the rounded value of X. Rounds X up or down to the nearest integer**
Trunc(X)	**Truncates X to the right of the decimal point**
Abs(X)	**Return the absolute value of X**

4. Read and write function

Read function:

readint(Var) : read integer variable.
Readchar(Var) : read character variable.
Readreal(Var) : read read (decimal) variable.
Readln(Var) : read string.

Write function

Write(Var) : write variable of any type.

Example 1: write prolog program to read integer value and print it.

```
Domains
    I = integer
Predicates
   print.
Clauses
       Print:- write (“please read integer number”), readint(X),
                write(“you read”,X).

Goal
   Print.
```

Output:
Please read integer number 4
You read 4

Example2: write prolog program that take two integer input us integer and print the greater.

```
Domains
    I = integer
Predicates
   Greater ( i,i)
Clauses
   Greater(X,Y):- X>Y,write(“the greater is”,X).
   Greater(X,Y):- write (“ the greater is “,Y).
Goal
   Greater(4,3).
```

Output:
The greater is 4

H.W:
1. write prolog program that read any phrase then print it.
2.write prolog program that read an integer number then print it after multiplying it by any other integer like 5.

Lecture five: More examples

This lecture present several example that intended to display various way to write prolog program, how to write if –else program ,divide problem into several parts then combine them in a single rule and how to write program describe specific problem.

Example 1: write prolog program to check if the given number is positive or negative.
Basic rule to check the number

```
If   X>=0   then
                X is positive
          Else
               X is negative
```

```
Domains
    I= integer
Predicates
    Pos_neg(i)
Clauses
    Pos_neg(X):-X>=0, write("positive number"),nl.
    Pos_neg(_ ):-write("negative number"),nl.

Goal
     Pos_neg(4)
```

Output:
    Positive number

Note: nl mean new line.

Example 2: write prolog program to check if a given number is odd or even.
Basic rule to check number

If X mod 2=0 then
X is even number
Else
X is odd number

Predicates
Odd_even(integer)
Clauses
Odd_even(X):-X mod 2= 0, write (“even number”), NL.
Odd_even(X):- write (“odd number”), nl.

Goal
Odd_even(5)

Output
Odd number

Example 3: write prolog program to combine both rule in example 1 and example2.

Domains
I= integer
Predicates
Pos_neg(i)
Odd_even(i)
Oe_pn(i)
Clauses
Oe_pn(X):-pos_neg(X),odd_even(X).
Odd_even(X):-X mod 2= 0, write(“ even number”),nl.
Odd_even(X):- write(“odd number”),nl.
Pos_neg(X):-X>=0, write(“positive number”),nl.
Pos_neg(_):-write(“negative number”),nl.

Goal
Oe_pn(3)

Output:
Odd number
Positive number

Note: the rule of same type must be gathering with each other.

Example 4 : write prolog program to describe the behavior of the logical And gate.

Truth table of And gate

X	Y	Z
0	0	0
1	0	0
0	1	0
1	1	1

Sol 1:

```
Domains
     I= integer
Predicates
   And1(I, I , I )

Clauses
   And1(0,0,0).
   And1(0,1,0).
   And1(1,0,0).
   And1(1,1,1).

Goal
   And1 (0,1,Z)
```

Output:
Z =0

Sol 2:
From the truth table we can infer the following rule:

```
If  X= Y then
        Z= X
    Else
        Z =0
```

```
Domains
    I= integer
Predicates
    And1 (I ,I, I )
Clauses
     And1 (X,Y,Z):- X=Y, Z=X.
     And1(X,Y,Z):- X<> Y, Z=0.

Goal
   And1(0,0,Z)

Output
   Z=0
```

H.W

1. Write prolog program that read character and check if it's a capital letter, small letter, digit or special character.

2. Modify prolog program in example 3 such that the value of X is read inside the program.

3. Write prolog program that describe the operation of logical Or gate.

Lecture six:

1. Cut and fail function

2. Negation

1. cut

Represented as "**!**" is a built in function always True , used to stop backtracking and can be placed any where in the rule, we list the cases where "!" can be inserted in the rule:

1 .R:-f1, f2,!. "f1, f2 will be deterministic to one solution.
2. R:-f1,!,f2. " f1 will be deterministic to one solution while f2 to all .
3. R:- !,f1,f2. "R will be deterministic to one solution.

Example1 : program with out use cut.

```
Domains
    I= integer
Predicates
   No( I )
Clauses
  No (5).
  No (7).
  No (10).

Goal
   No (X).
```

Output:

```
X=5
X=7
X=10
```

Example 2: program using cut.

```
Domains
    I= integer
Predicates
    No( I )
Clauses
    No (5):-!.
     No (7).
```

```
    No (10).
Goal
   No (X).

 Output:
      X=5.
```

Example3: program with out using cut.

```
Domains
      I =integer
      S = symbol
Predicates
     a (I )
     b (s )
     c ( I, s )
Clauses
     a(10).
     a(20)
     b(a)
     b(c)
      c (X, Y):- a (X), b (Y).

Goal
   c(X,Y).

Output:
      X= 10    Y=a
      X=10     Y=c

      X=20      Y=a
      X=20      Y=c
```

Example 4: using cut in the end of the rule.

```
Domains
      I =integer
      S = symbol
Predicates
     a(I )
     b (s )
```

```
   c (I, s )
Clauses
     a(10).
      a(20)
      b(a)
      b(c)
       c (X, Y):- a (X), b (Y),!.

Goal
     c(X,Y).
```

Output:

```
     X= 10     Y=a
```

Example 5: using cut in the middle of the rule.

```
Domains
     I =integer
     S = symbol
Predicates
    a(I )
    b (s )
    c ( I, s )
Clauses
     a(10).
     a(20)
     b(a)
     b(c)
     c (X, Y):- a (X),!, b (Y).

Goal
     c(X,Y).
```

Output:

```
     X= 10     Y=a
               Y=c
```

2. Fail

Built in function written as word “fail” used to enforce backtracking, place always in the end of rule, produce false and can be used with internal goal to produce all possible solution.

Example 6:

```
Predicates
      Student ( symbol , integer)
       Printout.
Clauses
     Student (aymen,95).
     Student(zainab,44).
     Student(ahmed,60).

     Printout:-student(N,M),write(N,”   “,M),nl,fail.

Goal
    Printout.
```

Output:

```
       aymen   95
       zainab    44
       ahmed    60
         No
```

Example 7:

```
Predicates
      Student ( symbol , integer)
       Printout.
Clauses
     Student (aymen,95).
      Student(zainab,44).
      Student(ahmed,60).
```

```
Printout:-student(N,M),write(N,"  ",M),nl,fail.
 Printout.

Goal
  Printout.

Output:
   aymen   95
   zainab    44
   ahmed    60
    Yes
```

3. Negation

Exceptions and return false in specific situation. Can be implemented using:

1. Cut-fail.
2. Not.

1. Cut-fail

Example 8:

Ahmed likes swimming and he want to visit all middle east seas accept the dead sea. Write prolog program to describe this situation.

A: using fail.

```
Predicates
    Visit (symbol)
    Middle_east (symbol)

Clauses
    Visit (Sea) :- middle_east (Sea).
     Middle_east (deadsea):- fail.
     Middle_east( redsea).
     Middle_east(arabsea).
```

Goal

1. Visit (deadsea)
2. Visit (W).

Output:

1. No

2. W= red_sea
   W=arab_sea

B: using cut- fail

```
Predicates
      Visit (symbol)
Clauses
       Visit (Sea) :- Sea=deadsea,!,fail.
        Visit (X):-middle_east(X).
         Middle_east(redsea).
         Middle_east(arabsea).
```

Example 9: ban like all animals but snake, write prolog program for this case.

```
Predicates
        Like( symbol, symbol)
        Snake(symbol)
        Animal(symbol)

Clauses
       Like(ban ,X):- animal(X),X=snake,!,fail.
       Like(ban,X):- animal(X).
       Animal(cat).
       Animal(bird).
       Animal(dog).
```

2. using not

For example 8: we can write it using not as follow.

```
Predicates
   Visit ( symbol)
   Middle_east(symbol).
Clauses
```

```
Visit ( X):- middle_east(X),not (X = deadsea).
Middle_east( redsea).
Middle_east(arabsea).
```

H.w:

1. Trace the following clauses and find the output:

a. clauses

```
reading:- readchar(Ch),writ(Ch),Ch='#'.
Reading.
```

b.clauses

```
Go.
Go:-go.
Reading:- go,readchar(Ch),write(Ch),Ch='#,!.
```

3. Use negation to define the different relation: diff(X,Y) which is true when X and Y are different numbers.

Lecture seven: repetition and recursion

1. Repetition
2. Recursion
2.1 Tail recursion
2.2 Non-tail recursion

1. Repetition

In prolog there is a constant formula to generate repetition; this technique can generate repetition for some operation until the stopping condition become true.

Example: prolog program read and write a number of characters continue until the input character equal to '#'.

```
Predicates
Repeat.
Typewriter.

Clauses
Repeat.
Repeat:-repeat.
Typewriter:-repeat,readchar(C),write(C ),nl,C='#',!.
```

2.Recursion

In addition to have rules that use other rules as part of their requirements, we can have rules that use themselves as part of their requirements.

This kind of rule called "recursive "because the relation ship in the conclusion appears again in the body of the rule, where the requirements are specified.

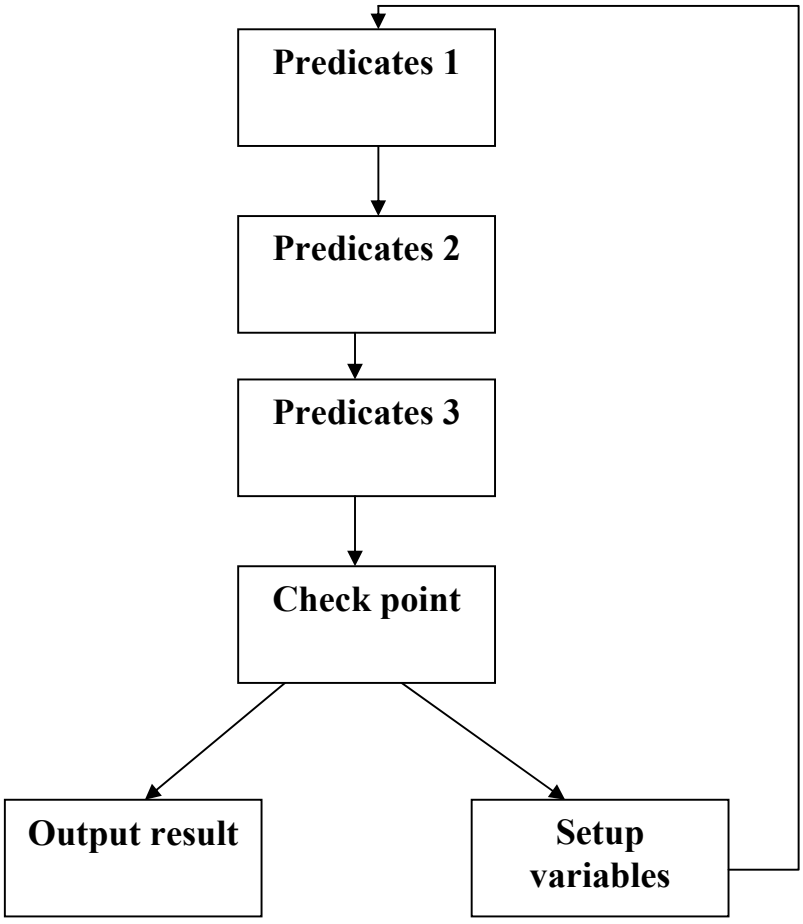
A recursive rule is a way of generating a chain of relationship for a recursive rule to be effective. However, there must be some place in the chain of relationship where the recursion stops.

This stopping condition must be answerable in the database like any other rule.

2.1 Tail Recursion

We place the predicate that cause the recursion in the tail of the rule as shown below:

Head :- p1,p2,p3, head.



Example 1: program to print number from n to 1.

```
Predicates
        A (integer)
Clauses
      A(1) :- write (1), nl ,!.
      A(M):- write (M) , nl, M1 = M -1, A( M1).

Goal
A(4)
```

Output:
4
3
2
1
Yes

Example 2: program to find factorial.

5! = 5*4*3*2*1

```
Predicates
     Fact ( integer, integer, integer)
Clauses
    Fact(1, F, F):-!.
    Fact(N,F,R):- F1=F*N , N1=N-1, fact(N1,F1,R).

Goal
Fact (5,1,F).
```

Output:
F = 120.

Example 3: program to find power .

$3^4 = 3*3*3*3$

```
Domains
   I= integer
Predicates
  Power ( I,I,I, I ).
Clauses
  Power ( X,Y,P,R):-  P1= P*X, Y1 =Y-1, power(X,Y1,P1,R).
  Power (_,0,P,P):-!.

Goal
Power(3,2,1,P)
```

Output
P= 9

2.2 Non –Tail Recursion (Stack Recursion)

This type of recursion us the stack to hold the value of the variables till the recursion is complete. The statement is self – repeated as many times as the number of items in the stack.. Below a simple comparison between tail and non-tail recursion.

Tail recursion	Non-tail recursion
1. Call for rule place in the end of the rule. 2. It is not fast as much as stack recursion. 3. Use more variable than stack recursion.	1. Call for the rule place in the middle in the rule. 2. Stack recursion is fast to implement. 3. Few parameters are used.

Example 4: factorial program using non-tail recursion.

```
Predicates
        fact(integer,integer).
Clauses
        fact(1,1).
        fact(N,F):- N>1,N1=N-1,fact(N1,F1),F=N*F1.
```

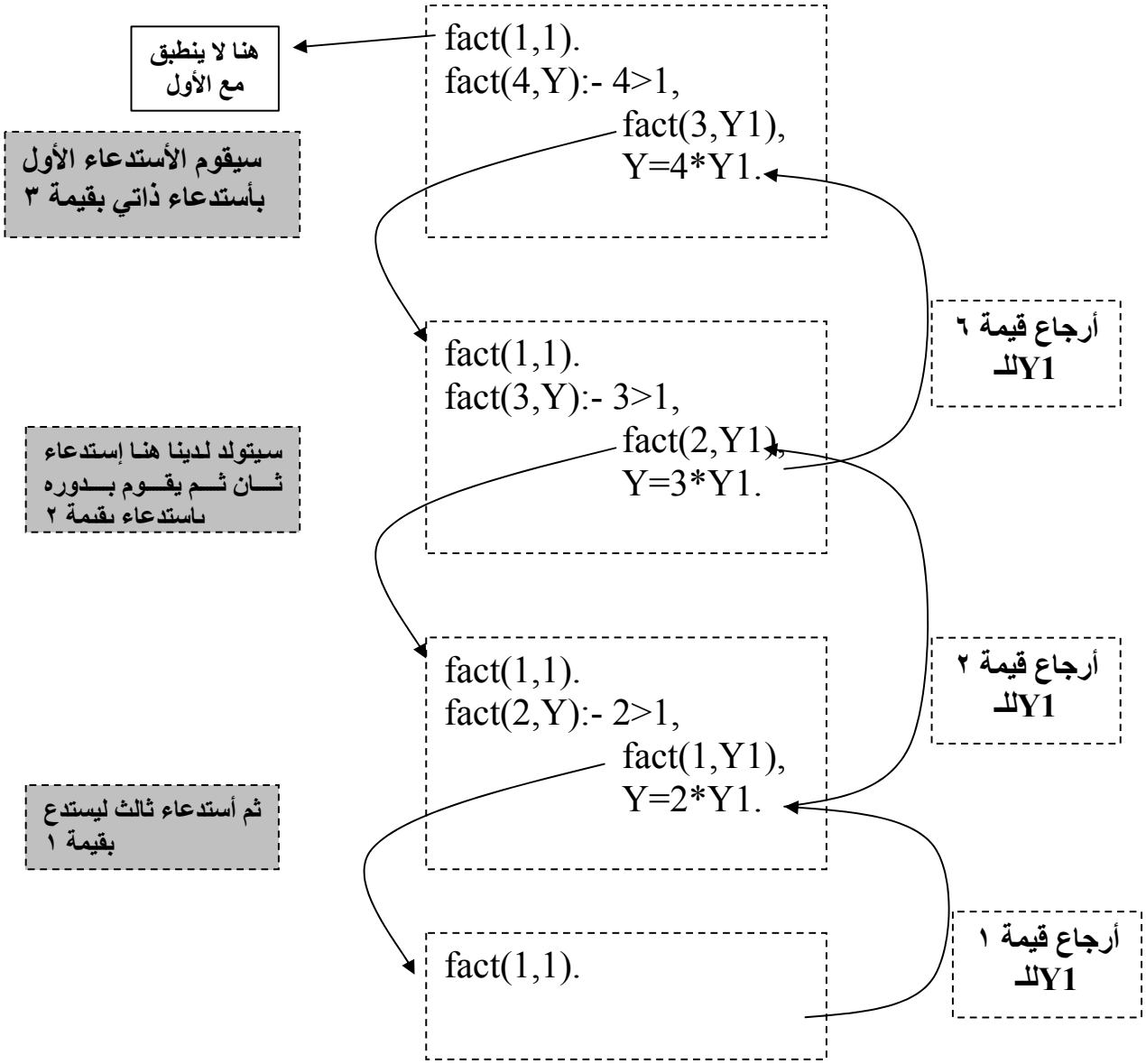
Goal
Fact (4,Y)

Output:
Y =24.

Example 5: power program using non-tail recursion.

```
Predicates
  Power (integer, integer, integer)
 Clauses
   Power (_,0,1):-!.
   Power ( X,Y,P) :- Y> 0, Y1=Y -1, power (X,Y1,P1),PZ= X*P1.
Goal
   Power (3,2,Z)
```

Output
Z = 9.



H.W

1. Write prolog program to find the sum of 10 integer element using tail and non tail recursion.
2. Write prolog program to find the maximum value between 10 elements.
3. Write prolog program to find the minimum value between 10 elements.
4. *Find the sum S = 1+2 + 3 ….+N*

Lecture eight:

String standard predicates

1. ***Isname(string) test if the content of the string is name or not***

Isname("abc") yes
Isname("123"). No

2.char_int(char,integer) convert the character to its integer value and the opposit

Char_int('A',X)
X=65
Char_int(X,65)
X='A'

3. Str_char(string,char) convert the string (of one char) to char and the opposit

Str_char("A",X)
X='A'
Str_char(X,'A')
X="A"

4. str_real (string,real) convert the string (ofreal) to real and the opposit

Str_real("0.5",X)
X=0.5
Str_real(X,0.5)
X="0.5"

5.Fronttoken(string,string,string).

Take token of word from the string and return the reminder of the string .

Fronttoken(string,token,rem).

Fronttoken("ab cd ef",X,Y).
X="ab" y="cd ef"
Fronttoken("c def",X,Y)
X="cd" Y="ef"

6. ***Frontstring(integer,string,string,string)***
   Take a string(str) with length specified by the integer value and return the reminder

Frontstring(integer,string,str,rem)

Frontstr(3,"ahmed",X,Y)
X="ahm" Y="ed"
Frontstr(2,"abcde",X,Y).
X="ab" Y="cde"
Frontstr(3,S,"ahm","ed").
S="ahmed"

7. ***Frontchar(string,char,string).***
   Take one char from a specific string and return the reminder

Frontchar(string,char,rem).
Frontchar("ahmed",X,Y)
X='a' Y="hmed"
Frontchar(X,'a',"hmed")
X="ahmed"

8. ***Str_len(string,length)***
   Return the length of specific string

Str_len("ahmed",X)
X=5
Str_len("ab",X)
X=2
Str_len("ab",3) no
Str_len(X,3) X="---"

9. ***Concat(string,string,string).***
   Concat two string together to produce one string

Concat("ab","cd",X)
X="abcd"

10. ***Upper_lower(string,string)***
    Convert the string in upper case(in capital letter) to the lower case (small letter) and the opposite.

Upper_lower(capital_letter,small_letter)
Upper_lower("ABC",X)

```
    X="abc"
Upper_lower("Abc",X)
    X="abc"
Upper_lower(,X,"abc")
     X="ABC"
```

Prolog Programs that deal with string

Ex1:Pogram that read two string and concat them in one string as upper case.

```
predicates
start(string)
clauses
start(X):-readln(S),readln(S1),concat(S,S1,S2),upper_lower(X,S2).
```

Goal
Start(X)

Output:
Ahmed
Ali
X=AHMEDALI yes

Ex2:program that read string of one character then return the integer value of this char.

```
predicates
start(integer).
clauses
start(X):-readln(S),str_char(S,X).
```

goal
start(X)

Output:
a
X=97
yes

Ex3: Program that take a string of words and print each word in a line as upper case.

```
predicates
start(string).
clauses

start(S):-fronttoken(S,S3,S2), upper_lower(S1,S3), write(S1),
nl,start(S2).
start("").
```

Goal
Start("ali is a good boy").

Output:
ALI
IS
A
GOOD
BOY
yes

Ex4: program that take a string and convert each character it contain to its corresponding integer value.

```
Predicates
start(string).
clauses

start(S):-fronttoken(S,S3,S2), char_int(S1,I), write(I), nl , start(S2).
start("").
```

Goal
Start("abc").

Output:
97
98
99
Yes

Ex5: program that return the number of names in a specific string.

```
predicates
start(string,INTEGER).
clauses

start(S,X):-fronttoken(S,S1,S2),isname(S1),X1=X+1,start(S2,X1).
start(S,X):-fronttoken(S,_,S2),start(S2,X).
start("",X):-write("the number of names is", X).
goal
start("ali has 2 cars").
```

Output:
The no. of names is 3
Yes

Ex6:program that split a specific string to small string with length 3 char.

```
predicates
start(string).
clauses
start("").
start(S):-str_len(S,I),  I MOD 3=0, frontstr(3,S,S1,S2), write(S1),
nl,start(S2).
start(S):-concat(S," ",S1),start(S1).

Goal
Start("abcdefg").
```

Output:
abc
def
g
yes

H.W

1- *Write a prolog program that do the following: convert the string such as "abcdef" to 65 66 67 68 69 70.*

2-*Program tofind the number of tokens and the number of character in a specific string such as: "ab c def" the output is tokens and 6 character.*

Lecture nine:

1. list in prolog

2. syntax of list

3. head and tail

1. list in prolog

In prolog, a list is an object that contains an arbitrary number of other objects within it. Lists correspond roughly to array in other languages but unlike array, a list dose not require you to how big it will be before use it.

2. syntax of list

List always defined in the domains section of the program as follow:

Domains

list = integer*

- '*' refer to list object which can be of length zero or un defined.
- The type of element list can be of any standard defined data type like integer, char … ect or user defined data type explained later.
- List element surrounded with square brackets and separated by comma as follow: l = [1, 2, 3, 4].
- List consist of two parts head and tail , the head represent the first element in the list and the tail represent the remainder (i.e head is an element but tail is a list) . for the following list :

L = [1,2,3]

H = 1 T =[2,3]

H =2 T =[3]

H =3 T=[ ]

[ ] refer to empty list.

List can be written as [H|T] in the program, if the list is non empty then this statement decompos the list into Head and tail otherwise (if the list is empty) this statement add element to the list.

4. list and recursion

As maintained previous list consist of many element, therefore to manipulate each element in the list we need recursive call to the list until it become empty.

Example 1: program to print list element in one line.

```
Domains
   L = integer*
Predicates
  Print (L)
Clauses
  Print ([ ]):-!.
  Print ( [ H|T]):- write (H) , print (T).

Goal
   Print ([1,4,6,8]).
Output:
    1468
```

Example 2: program to find sum of integer list.

```
Domains
 I= integer
L=i*

Predicates
  Sum ( L I, I)
Clauses

Sum ( [ ],S,S):-!.
Sum( [H| T],S1,S):- S2 = S1+H , Sum (T,S2,S).

Goal
Sum ( [ 1,4,6,9],0 ,S).

Output
S = 20
```

Example 3: prolog program to spilt list into to list positive and negative list.

```
Domains
L= integer*

Predicates
Spilt ( L,L,L)

Clauses

Spilt ( [ ],[ ],[ ]):-!.
Spilt ( [ H| T],[H|T1],L2):- H>= 0,!,spilt (T, T1,L2).
Spilt ([H|T],L1,[H|T2] ) :- spilt ( T,L1,T2).

Goal
Spilt ([-1,4,-9,8,0],L1,L2).
L1 = [4,9,0]
L2 = [-1,-9]
```

H.W

1. Write prolog program to find the union of two lists.
2. Write prolog program to find the intersection between two lists.
3. Write prolog program to find the difference between two lists.
4. Write prolog program that check the equality between two lists.
5. Write prolog program to find the last element in a list.
6. Write prolog program to find the union of two lists.
7. Write prolog program to find the length of a list.
8. Write prolog program to find the index of specified element in a list.
9. Write prolog program to get the element at nth index lists.
10. Write prolog program that replace specified element in a list with value 0.
11. Write prolog program that delete a specified element in a lists.
12. Write prolog program that take two lists as input and produce a third list as output, this list is the sum of the two lists.
13. Write prolog program that multiply each element in the list by 5.
14. Write prolog program that sort a list descending.
15. Write prolog program that convert any given decimal number to its binary representation and store it in a list.

Lecture ten:

1) Data – Driven and Goal Driven Search (Reasoning Search):-

In data –driven search , sometimes called **Forward Chaining (F.W)** , the problem solver begins with the give facts and a set of rules for changing the state. Search proceeds by apply rules to facts to produce new facts . This process continues until it generates a path that satisfies the goal.

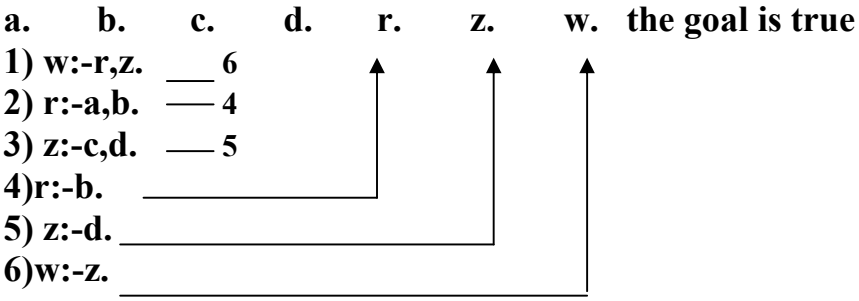
In Goal-Driven search , sometimes called **Backward Chaining (B.W)** , the problem solver begins with the goal to be used to generate this goal and determine what conditions must be true to use them. These conditions become the new goals, sub goals , for the search. This process continues , working backward through successive sub goals, until a path is generated that lead back to facts of the problem.

<u>Example of Data Driven Search (F.W)</u>

Using (F.W) to find if the goal w is true or false

a. b. c. d.
w:-r,z.
r:-a,b.
z:-c,d.

sol/



ملاحظات عن F.W

١) نبدأ بالحقائق (Facts) بالبحث عنها في القواعد(Rules) في جهة اليمين فأذا كانت موجودة تحذف من الطرف اليمين:-

٢) اذا أصبحت الrule في الطرف الأيمن فارغة تتحول ال rule الى fact والا فتضاف rule جديدة بعد عملية الحذف.

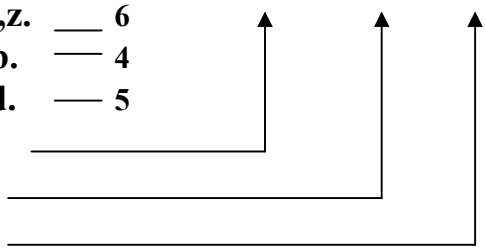
٣) بعد تأشير كل facts إذا كان الهدف موجود ضمن الfacts فهو true والا فهو false.

Example of Goal Driven Search (B.W)

Try the previous facts & rules to prove if (w) is true or false.

a. b. c. d. z. r. w. the goal is true

1) w:-r,z. ___ 6
2)z:-a,b. ___ 4
3)r:-c,d. ___ 5
4)r:-b.
5) r:-d.
6)w:-r.



ملاحظات عن F.W

١) نبدأ بالهدف وذلك بالبحث عنه في ال facts إذا كان موجود فهو true وألا فهو False.

٢) نبحث عن الهدف في Rules في جهة اليسار في حالة وجوده نحاول إثبات الطرف الأيمن True.

Example: Try the following facts & Rules with (F.W) & (B.W) Chaining.

a(1). B. c. d(1).

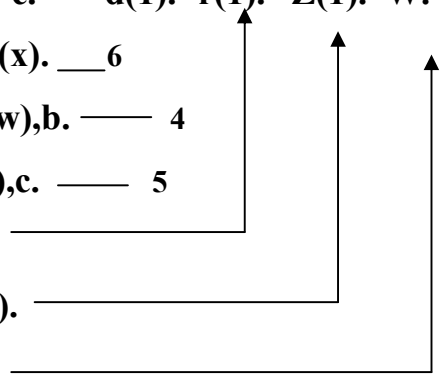
W:-r(x),z(x).

R(w):-a(w),b.

Z(v),c.

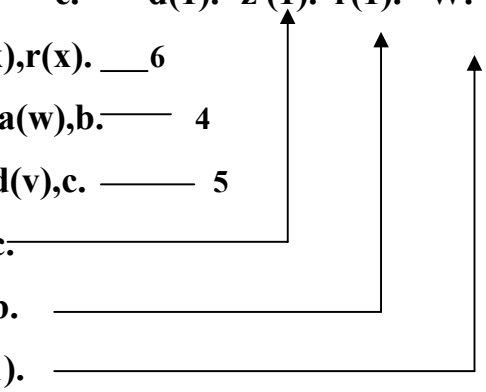
Sol/

a(1). B. c. d(1). r(1). Z(1). W. the goal is true



Sol/ B.W Chaining

a(1). B. c. d(1). z (1). r(1). W. the goal is true



H.W/ Using B.W & F.W chaining to reasoning that the goal (Z) is true or not.

a(1). b(2). c(3). d(1). e.

r:-a(x),b(y).

z:-e, not (f),not(b(3)),w.

w:-c(z),d(l),not (a(3)),r.

Lecture eleven:

Knowledge Representation

There are many methods can be used for knowledge representation and they can be described as follows:-

1- Semantic net.
2- Conceptual graph.
3- Frames
4- Predicates logic.
5- Clause forms

1) Semantic Net

It is consist of a set of nodes and arcs , each node is represented as a rectangle to describe the objects, the concepts and the events. The arcs are used to connect the nodes and they divided to three parts:-

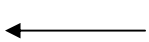
Is a: ⟶ for objects & types

Is ⟶ To define the object or describe it

Has a ⟶

can ⟶

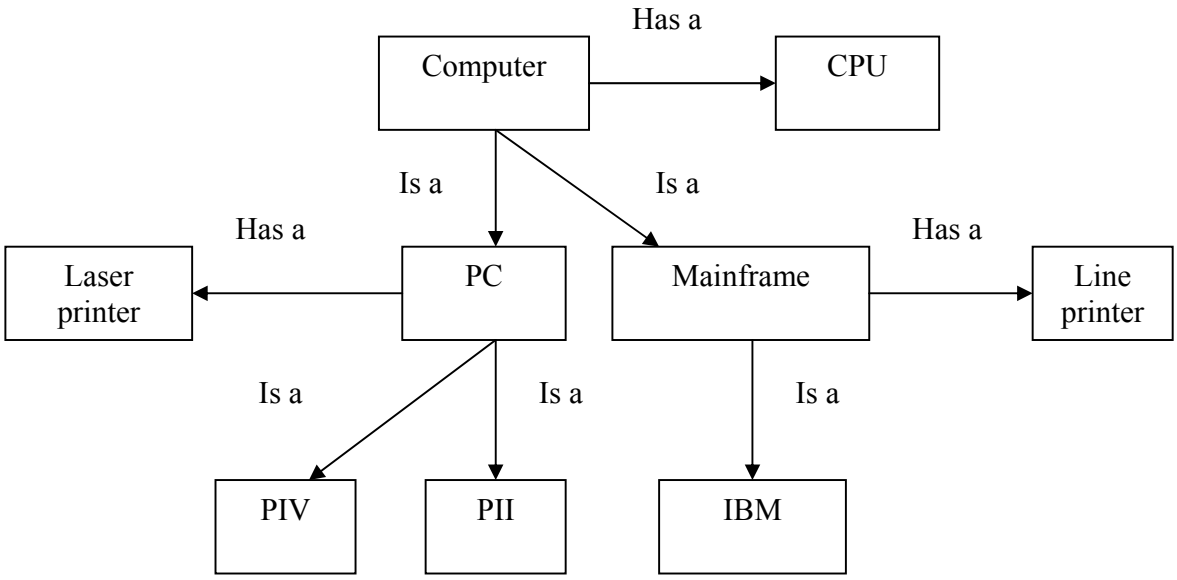
لتمثيل الأفعال والأحداث والكائنات



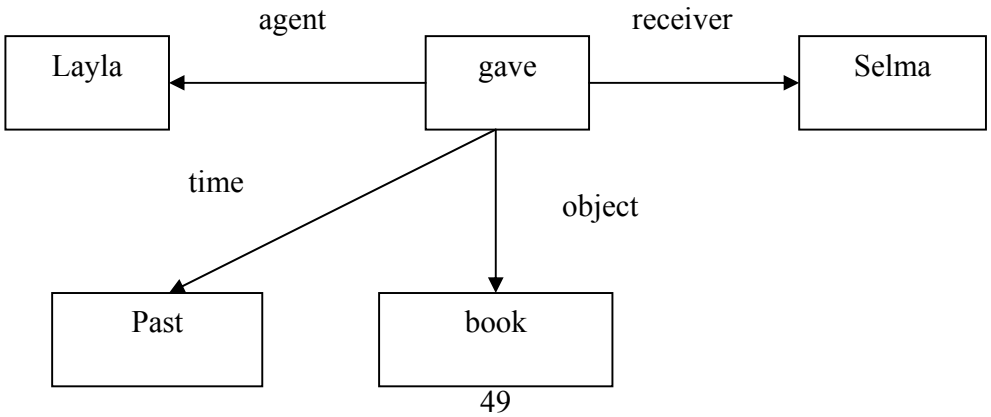
لتمثيل العلاقة بين الكائنات

في وصف اللغات الطبيعية فان arcs تخرج من الفعل لتوضح او لتشير الى الفاعل (agent) والمستقبل (Reciever) والكائن (object) كما تشير الى وقت حدوث الفعل أي في الماضي ، الحاضر او المستقبل.

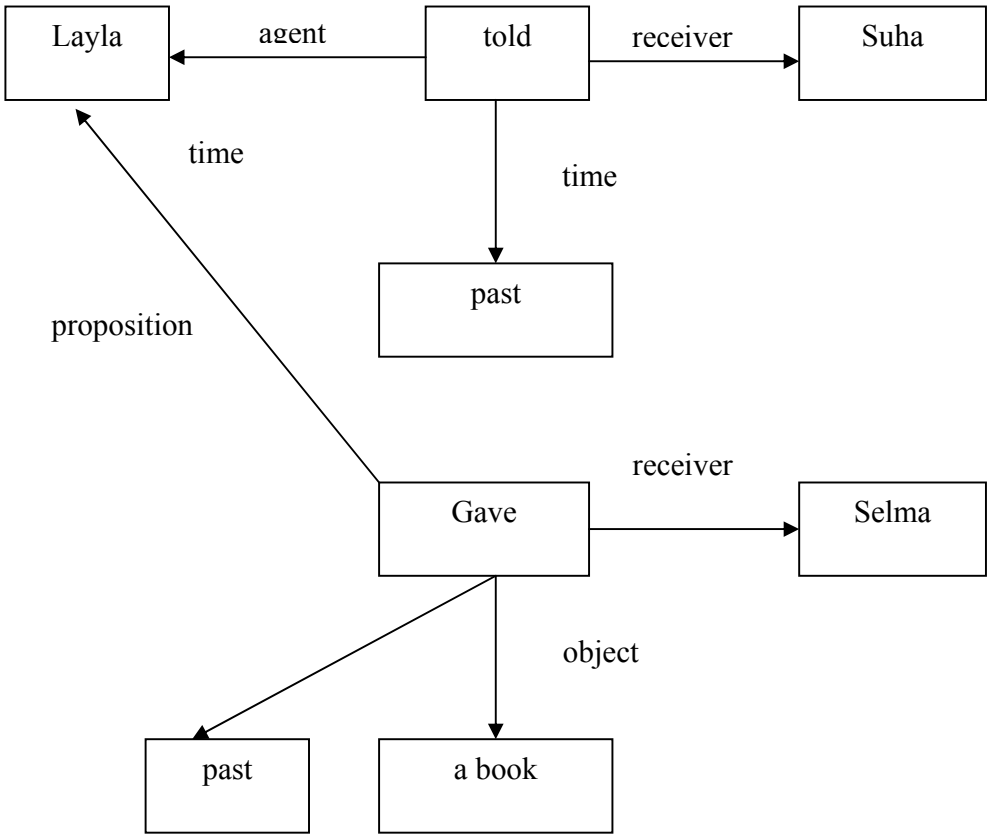
Example1: Computer has many part like a CPU and the computer divided into two type, the first one is the mainframe and the second is the personal computer ,Mainframe has line printer with large sheet but the personal computer has laser printer , IBM as example to the mainframe and PIII and PIV as example to the personal computer.



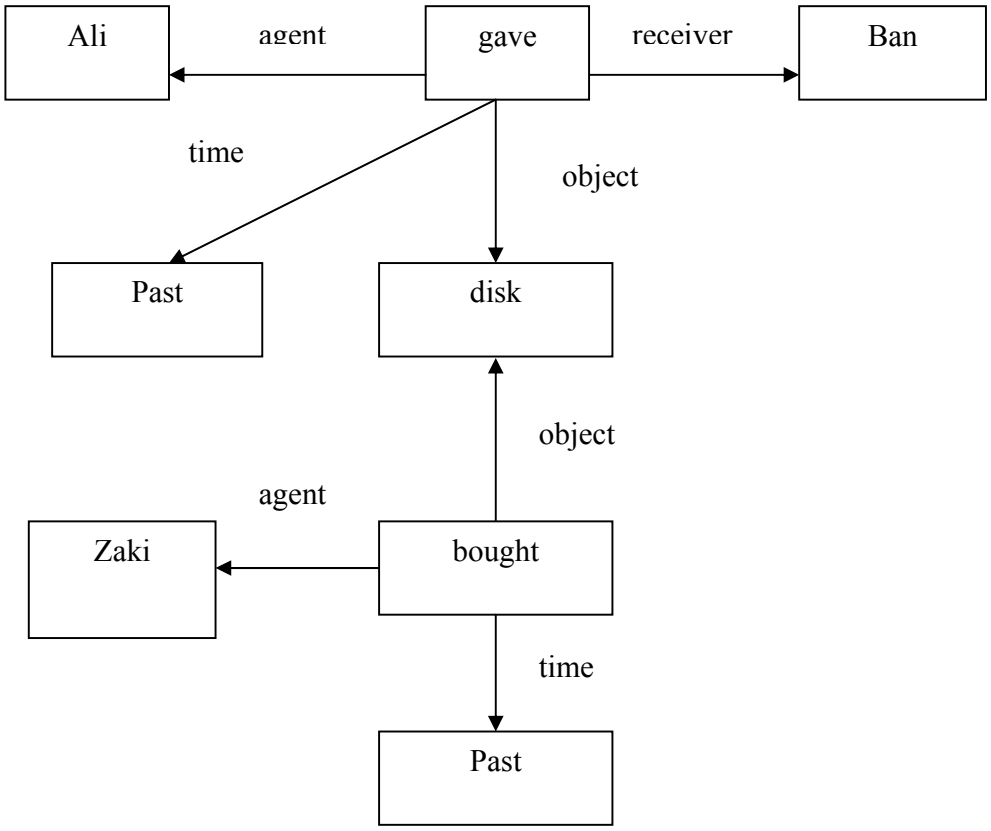
Example 2: Layla gave Selma a book



Example 3: Layla told Suha that she gave Selma a book



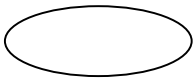
Example 4: Ali gave Ban a disk which is Zaki bought



2) The Conceptual Graph

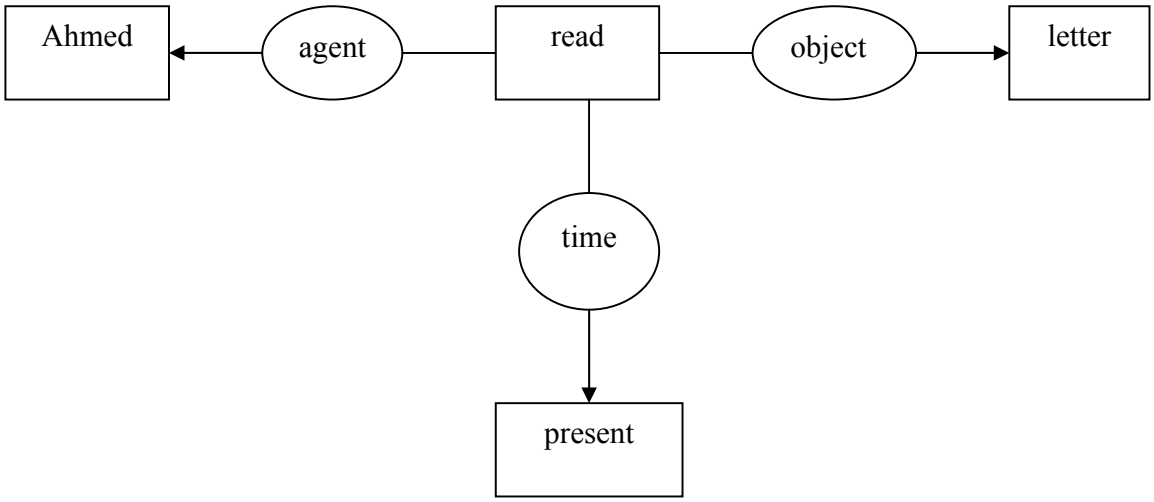
وهي طريقة لتمثيل المعرفة مشابهة لطريقة Semantic Net وتتكون من جزئيين:

يستخدم لتمثيل الأسماء والصفات والأفعال والثوابت

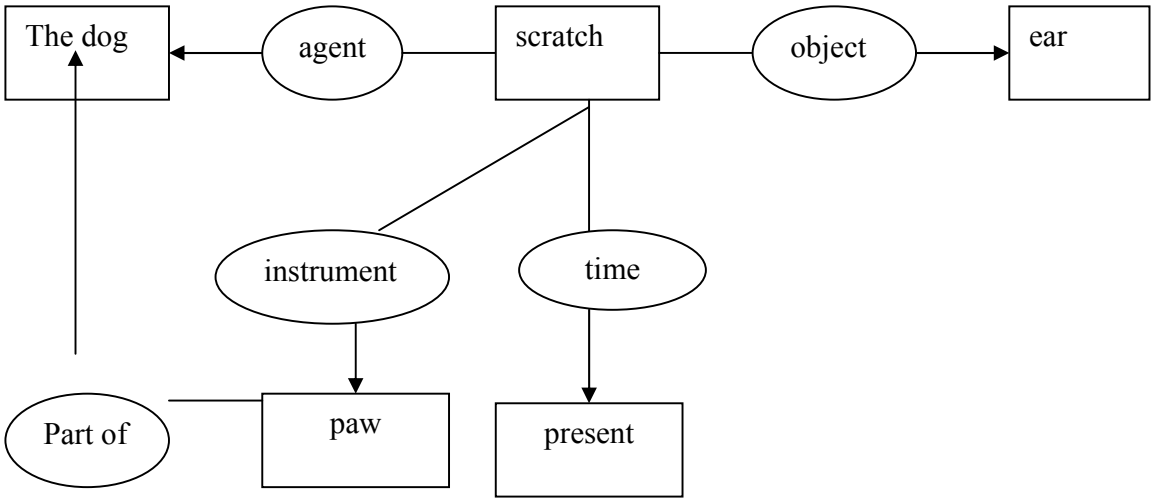


يستخدم لتمثيل أدوات التعريف والعلاقات

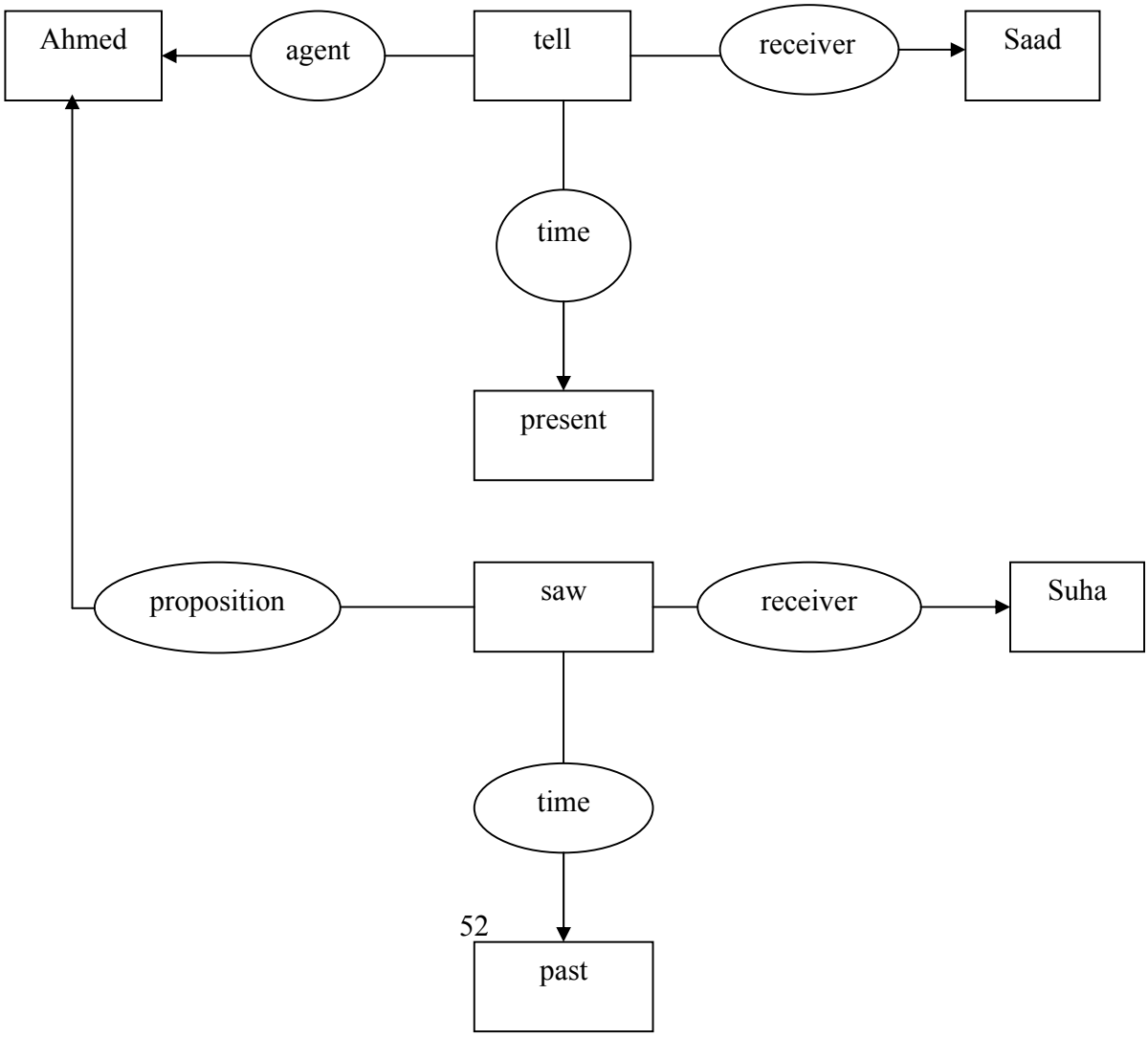
Example 1: Ahmed read a letter Yesterday



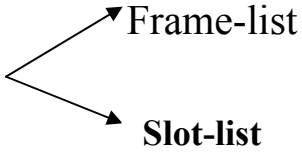
Example 2:- The dog Scratch it ear with is paw



Example 3: Ahmed tell Saad that he saw Suha



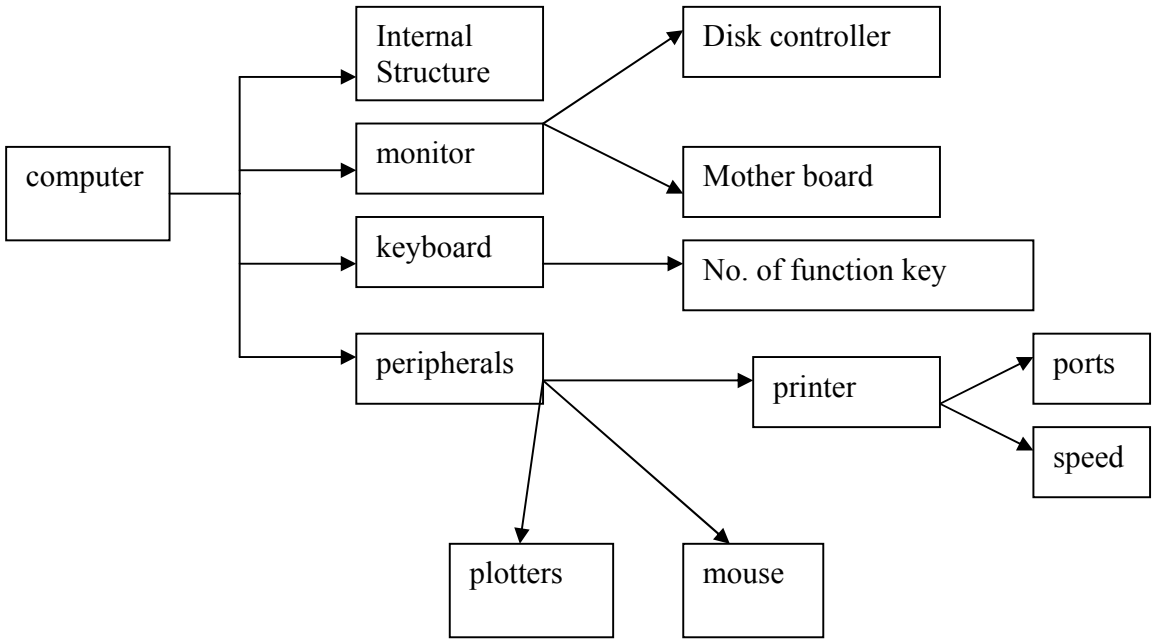
3) Frame:



Frame-list(node-name, parent, [child]).

Slot-list(node-name, parent).

Example:



Frame –list(computer,_ ,[Internal structure, monitor, keyboard , plotters]).

Frame-list(Internal structure, computer, [disk controller, mother board]).

Frame- list(printer, peripheral, [speed, ports]).

Slot-list(motherboard, Internal structure).

Slot-list(mouse, peripheral).

Homework 1: **solve with Semantic net**

Ships are divided in two types, the first is “Ocean lines” and the second is “Oil tank” , the ships has an engine , the oil tank are specified to transfer oil therefore it has “ fire tools” , the ocean lines are specified to transfer the traveler therefore it has “ swimming poot” , Ibnkaldon as an example to oil tank and ship b and ship n as an example to ocean line.

Homework 2: Using Semantic Net and Conceptual graph to solve the following statement:

1) Suha send a book to Tom.
2) Tom believe that Mustafa like cheese.
3) Monkey ema grasp the banana with hand.

Lecture twelve:

Search Algorithms:

To successfully design and implement search algorithms, a programmer must be able to analyze and predict their behavior.
Many questions needed to be answered by the algorithm these include:

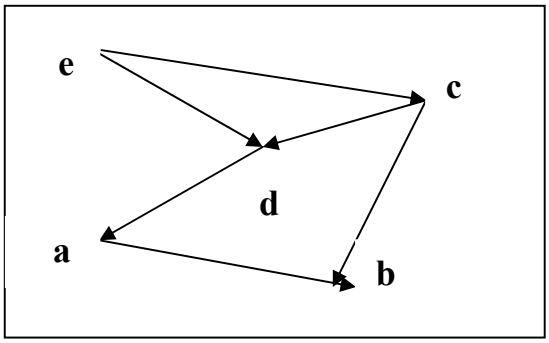
- is the problem solver guranteed to find a solution?
- Will the problem solver always terminate , or can it become caught in an infinite loop?
- When a solution is found , is it guaranteed to be optimal?
- -What is the complexity of the search process in terms of time usage ? space search?
- How can the interpreter be designed to most effectively utilize a representation language?

-State Space Search

The theory of state space search is our primary tool for answering these questions , by representing a problem as state space graph, we can use graph theory to analyze the structure and complexity of both the problem and procedures used to solve it.

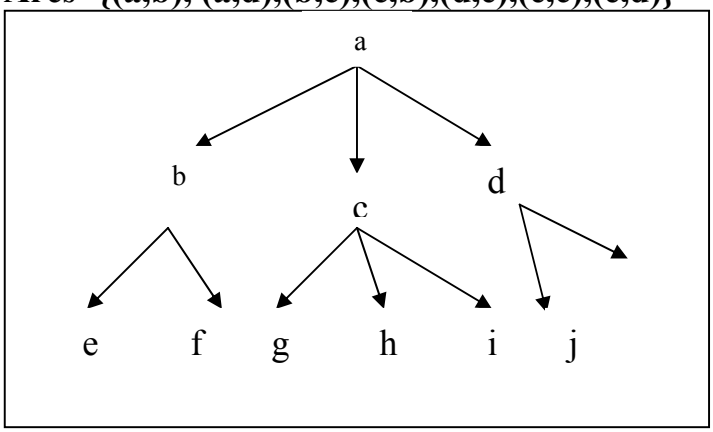
- **Graph Theory:-**
  A graph consist of a set of a nodes and a set of arcs or links connecting pairs of nodes. The domain of state space search , the nodes are interpreted to be stated in problem_solving process, and the arcs are taken to be transitions between states.

  Graph theory is our best tool for reasoning about the structure of objects and relations.



Nodes={a,b,c,d,e}

Arcs={(a,b), (a,d),(b,c),(c,b),(d,e),(e,c),(e,d)}



Nodes=={a,b,c,d,e,f,g,h,i}
Arcs={(a,b),(a,c),(a,d),(b,e),(b,f),(c,f),(c,g),(c,h),(c,i),(d,j)}

State Space Representation of Problems:-

A state space is represented by four_tuple [N,A,S,G,D], where:-

- **N** is a set of nodes or states of the graph. These correspond to the states in a problem –solving process.
- **A** is the set of arcs between the nodes. These correspond to the steps in a problem –solving process.
- **S** a nonempty subset of N ,contains the start state of the problem.
- **GD** a nonempty subset of N contains the goal state of the problem.

A solution path:- Is a path through this graph from a node S to a node in GD.

State Space Searchs examples:-

1) Monkey and Banana Problem

There is a monkey at the door in to a room. In the middle of the room a banana is hunging from the ceiling. The monkey is hungry and wants to get the banana , but he cannot stretch high enough from the floor. At the window of the room there is a box the monkey may use.

The monkey can perform the following actions:-

- walk on the floor
- Climb the box
- Push the box a round (if it is already at the box).
- Grasp the banana if standing on the box directly under the banana.

The question is (Can the monkey get the banana?), the initial state of the world is setermind by:-

1- Monkey is at door.
2- Monkey is on floor.
3- Box is at Window.
4- Monkey does not have banana

Initial state :- State (at door, on floor, at window, has not).

At door ⟶ horizontal position of monkey

On floor ⟶ vertical position of monkey

At window ⟶ Position of box

Has not ⟶ monkey has not banana

Goal state:-State (_,_,_,has).

State1 ⟶ state2

Move (state1, move, state2).

State1: is the state before the move.

Move: is the move executed.

State2:is the state after the move.

To answer the question :- Can the monkey in some initial state (state) get the banana?

This can be formulated as a predicate canget(state).The program canget can be based on two observation:-

1) The program:- for any state in which the monkey already has the banana. The predicate canget must certainly be true , no move is needed in this case:

Canget(state(state(_,_,_,has)).

2) In other cases one or more moves are necessary.

Canget (state):-move (state1,move,state2),canget (state2).

A program of monkey and banana problem:-

Move (state (at door , on floor , at window , has not), walk, state (at box , on floor , at window , has not)).

Move (state (at box , on floor , at window , has not), push , state (middle, on floor, middle, has not)).

Move (state (middle, on floor, middle, has not), climb, state (middle, on box, middle, has not)).

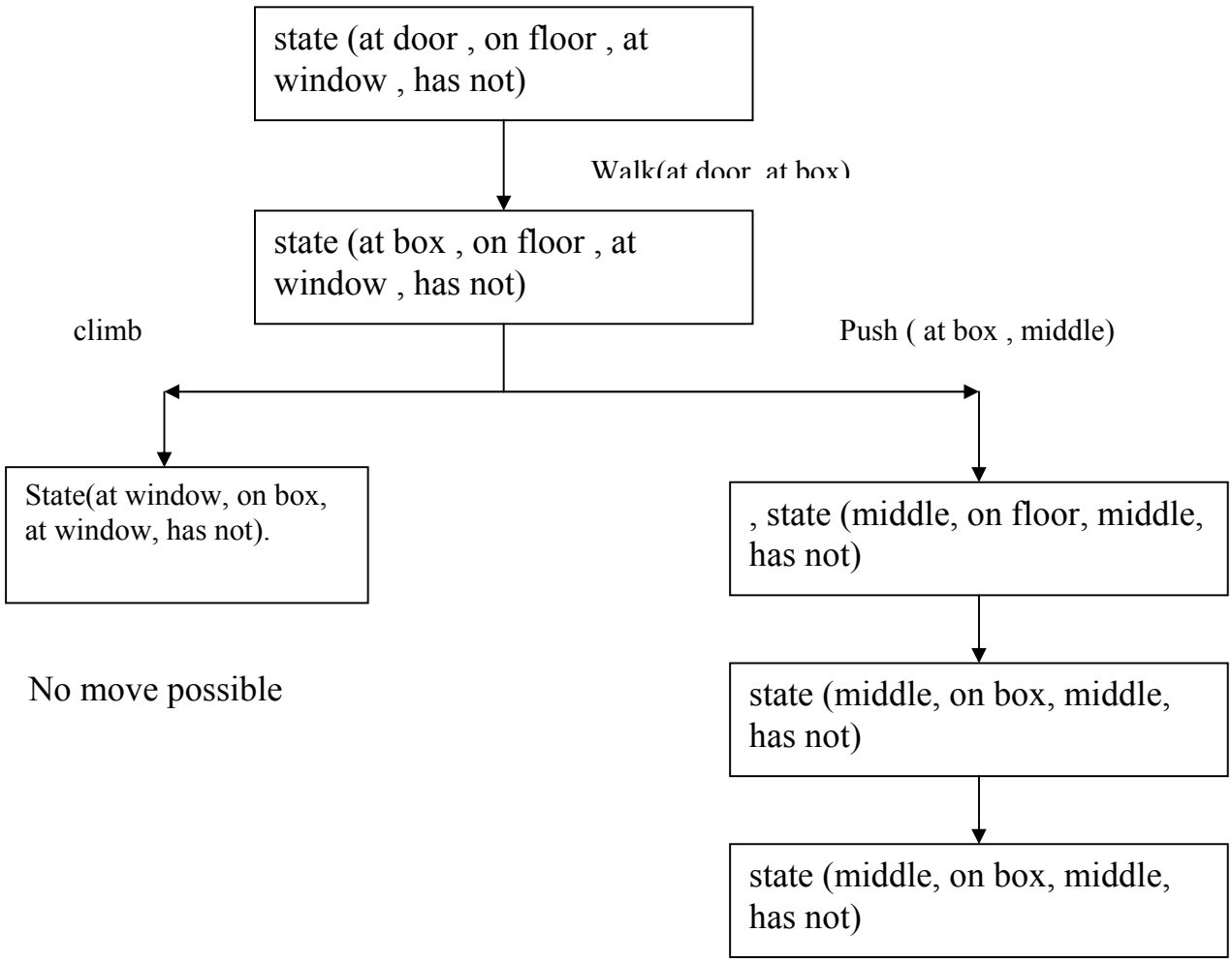
Move (state (middle, on box, middle, has not),grasp, state (middle, on box, middle, has not)).

Canget(state(_,_,_,has)).

Canget (State1):- move (state1,move,state2), canget (state2).

Goal= canget (state(at door, on floor, at window ,has not)).

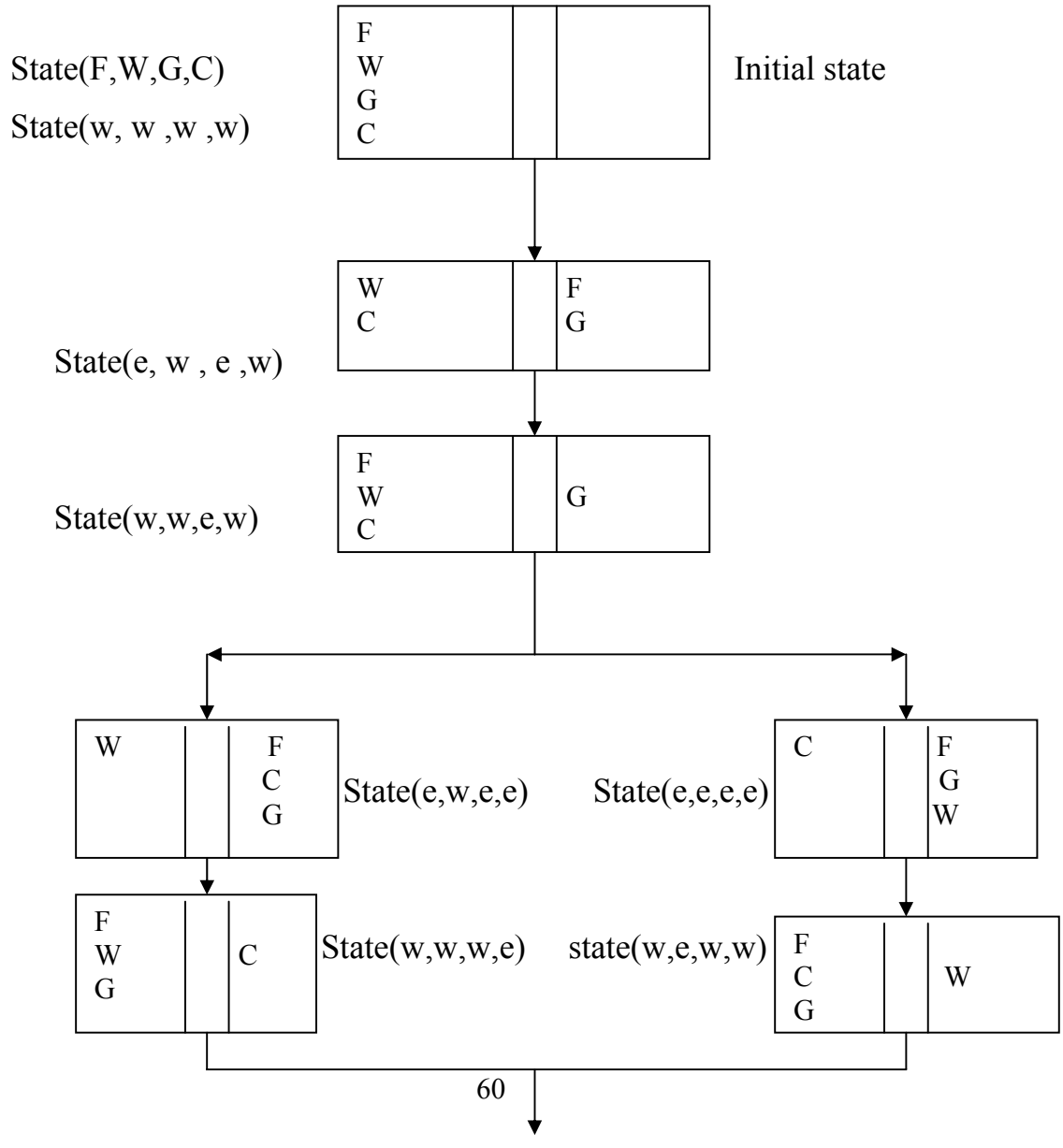
- The monkey and banana problem can be represented by the following state space:-

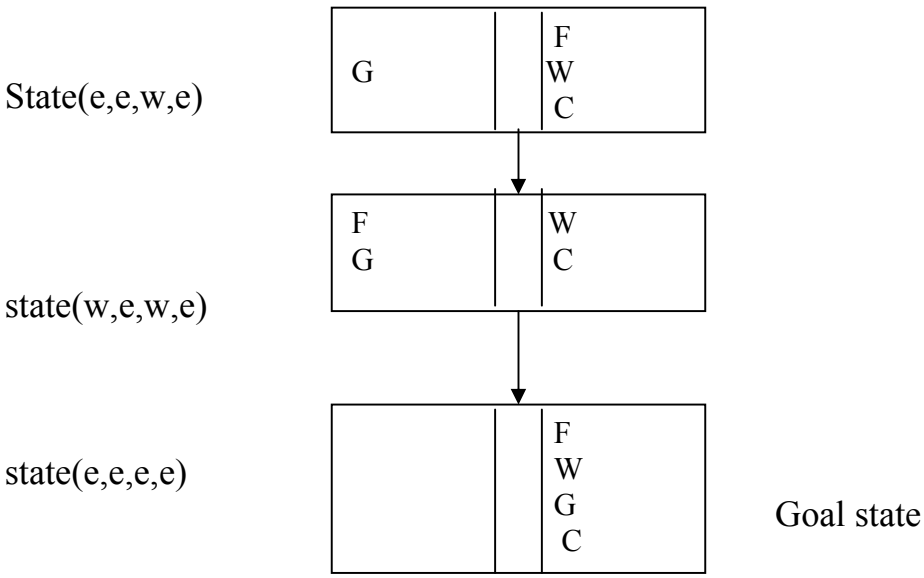


Lecture thirteen:

2) The Farmer , Wolf, Goat and Cabbage Problem:-

A farmer wants to move himself , a wolf , a goat and some cabbage across a river. Unfortunately his boat is soting , the farmer can take only one of his possession across any trip worse yet, an attended wolf will eat a goat, and and attended gaot will eat cabbage , so the farmer not leave the wolf alone with goat or the goat alone with the cabbage. What he is to do?





The following move rule operates only when the farmer and wolf are in the same location and takes them to the opposite side of the river. Note that the goat and cabbage do not change their present location:-

Move(state(X,X,G,C),state(Y,Y,G,C)):-opp(X,Y).

X opposite (opp) the value of Y

Opp (e , w).

Opp (w, e).

A predicate must be created to test whether each new state is safe , so that nothing is eaten in the process of getting across the river. These unsafe situations may be represented with the following rules:-

Unsafe (state(X,X,Y,Y)):-opp(X,Y).

Unsafe(state(X,W,Y,Y)):-opp(X,Y).

Now, a not unsafe test must be added to move rule:-

Move(state (X,X,G,C),state(Y,Y,G,C)):-opp(X,Y), not (unsafe (state (Y,Y,G,C))).

3) Water Jug Problem

You are give two jugs , a 4-gallon one and a 3-gallon one. Neither has any measuring markers on it. There is a pump that can be used to fillthe jugs with water. How can you get exactly 2 gallons of water in to the 4-gallon jug?

The state space for this problem can be described as the set of ordered pairs of integers (x,y) , such that x=0,1,2,3 or 4 and y=0,1,2, or 3; x represent the number of gallons of a water in the 4-gallon jug , and y represents the quantity of water in 3-gallon jug. The start state is (0,0). The goal state is (2,n) for any value of n (since the problem does not specify how many gallons need to be in the 3-gallon jug).

1) (X,Y: X<4) ⟶ (4,Y) Fill the 4 – gallon jug

2) (X,Y: Y<3) ⟶ (X,3) Fill the 3-gallon jug

3) (X,Y:X>0) ⟶ (X-D,Y) Pour some water out of the 4- gallon jug

4) (X,Y:X>0) ⟶ (X,Y-D) Pour some water out of the 3- gallon jug

5) (X,Y:X>0) ⟶ (0,Y) Empty the 4-gallon jug on the ground

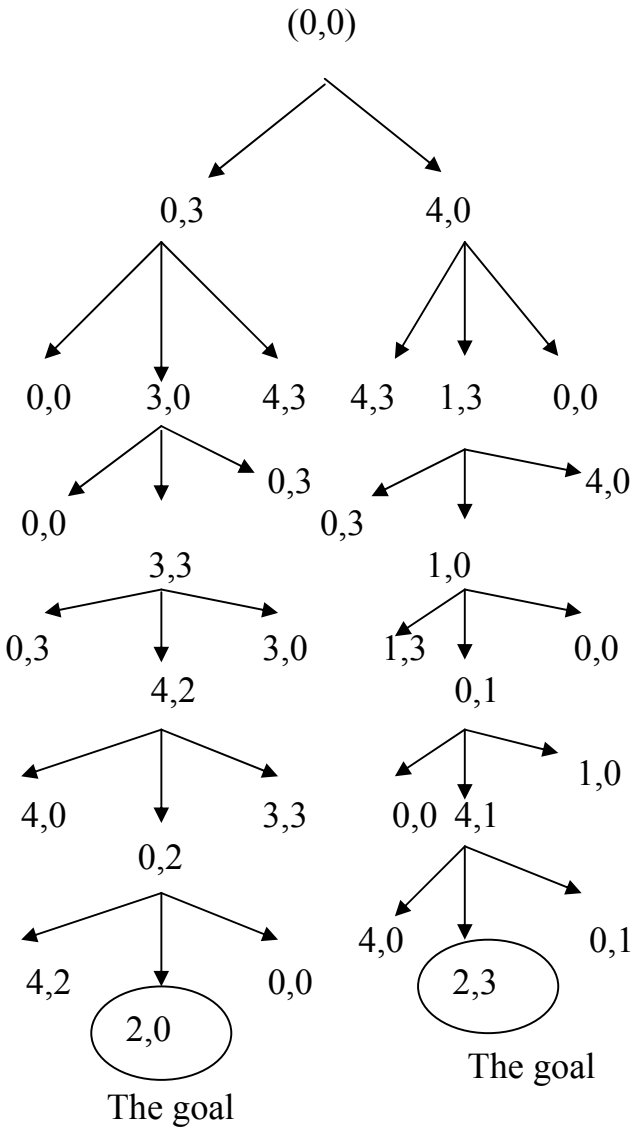
6) (X,Y: Y>0) ⟶ (X,0) Empty the 3-gallon jug on the ground

7) (X,Y: X+Y>=4 ^ Y>0) ⟶ (4,Y-(4-X)) pour water from the 3-gallon jug into the 4-gallon jug until the 4-gallon jug is full.

8) (X,Y:X+Y<=4 ^ X>0) ⟶ (X-(3-Y),3) pour water from the 4-gallon jug into the 3-gallon jug until the 3-gallon jug is full.

9) (X,Y:X+Y<=4 ^ Y>0) ⟶ (X+Y,0) pour all the water from 3-gallon jug into the 4-gallon jug.

10) (X,Y: X+Y<=3 ^ X>0) ⟶ (0,X+Y) pour all the water from 4-gallon jug into the 3-gallon jug.



The solutions of water jug problem

Lecture fourteen:

Blind Search

This type of search takes all nodes of tree in specific order until it reaches to goal. The order can be in breath and the strategy will be called breadth – first – search, or in depth and the strategy will be called depth first search.

Breadth – First – Search

In breadth –first search , when a state is examined , all of its siblings are examined before any of its children. The space is searched level-by-level , proceeding all the way across one level before doing down to the next level.

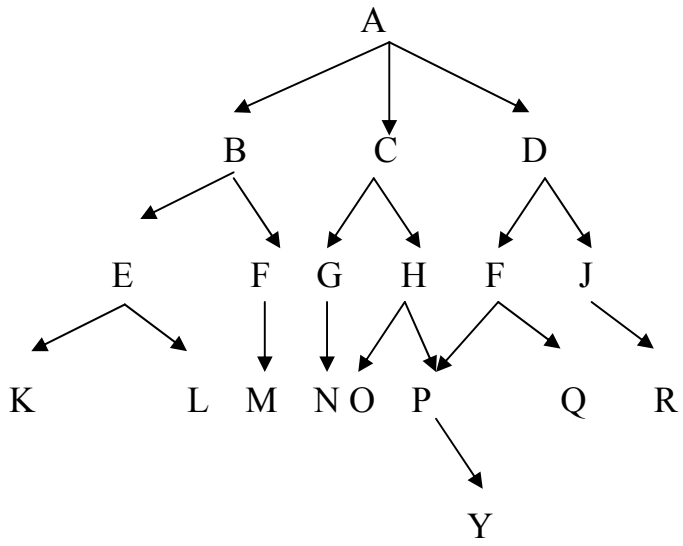


Fig (2 – 1): Breadth – first – search

1 – Open = [A]; closed = [ ].

2 – Open = [B, C, D]; closed = [A].

3 – Open = [C, D, E, F]; closed = [B, A].

4 – Open = [D, E, F, G, H]; closed = [C, B, A].

5 – Open = [E, F, G, H, I, J]; closed = [D, C, B, A].

6 – Open = [F, G, H, I, J, K, L]; closed = [E, D, C, B, A].

7 – Open = [G, H, I, J, K, L, M]; closed = [F, E, D, C, B, A].

8 – Open = [H, I, J, K, L, M, N,]; closed = [G, F, E, D, C, B, A].

9 – and so on until either U is found or open = [ ].

Depth – first – search

In depth – first – search, when a state is examined, all of its children and their descendants are examined before any of its siblings.

Depth – first search goes deeper in to the search space when ever this is possible only when no further descendants of a state cam found owe its

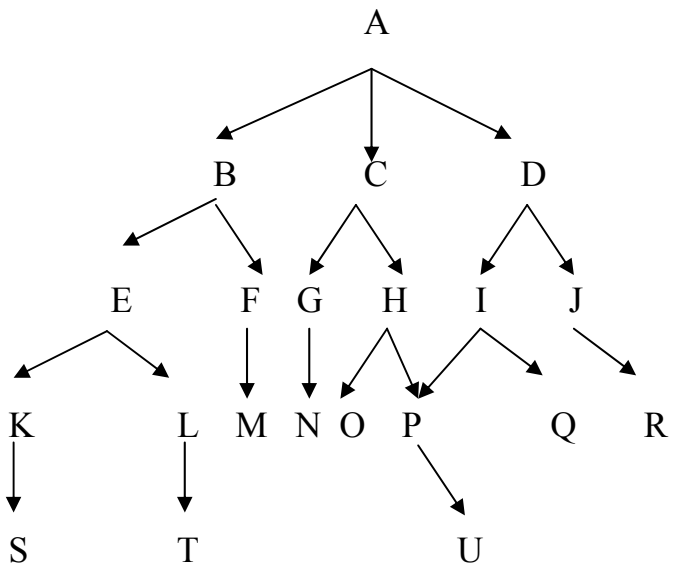


Fig (2 – 2) Depth first search

1 – Open = [A]; closed = [ ].

2 – Open = [B, C, D]; closed = [A].

3 – Open = [E, F, C, D]; closed = [B, A].

4 – Open = [K, L, F, , D]; closed = [E, B, A].

5 – Open = [S, L, F, C, D]; closed = [K, E, B, A].

6 – Open = [L, F, C, D]; closed = [S, K, E, B, A].

7 – Open = [T, F, C, D]; closed = [L, S, K, E, B, A].

8 – Open = [F, C, D,]; closed = [T, L, S, K, E, B, A].

9 – Open = [M, C, D] as L is already on; closed = [F, T, L, S, K, E, B, A].