PROLOG DATABASE

- A Logic Database is a set of facts and rules (i.e. a logic program):

```
father_of(john,peter) <-.
father_of(john,mary) <-.
father_of(peter,michael) <-.

mother_of(mary, david) <-.
```

```
grandfather_of(L,M) <- father_of(L,N),
                       father_of(N,M).
grandfather_of(X,Y) <- father_of(X,Z),
                       mother_of(Z,Y).
```

- Given such database, a logic programming system can answer questions (queries) such as:

```
<- father_of(john, peter).
Answer:  Yes
<- father_of(john, david).
Answer:  No
<- father_of(john, X).
Answer:  {X = peter}
Answer:  {X = mary}
```

```
<- grandfather_of(X, michael).
Answer:  {X = john}
<- grandfather_of(X, Y).
Answer:  {X = john, Y = michael}
Answer:  {X = john, Y = david}
<- grandfather_of(X, X).
Answer:  No
```

- Rules for `grandmother_of(X, Y)`?

Logic Programs and the Relational DB Model

Traditional →	*Codd's Relational Model*	
File	Relation	Table
Record	Tuple	Row
Field	Attribute	Column

- *Example*:

Name	Age	Sex
Brown	20	M
Jones	21	F
Smith	36	M

Person

Name	Town	Years
Brown	London	15
Brown	York	5
Jones	Paris	21
Smith	Brussels	15
Smith	Santander	5

Lived–in

- The order of the rows is immaterial.
- (Duplicate rows are not allowed)

Relational Database	→	*Logic Programming*
Relation Name	→	Predicate symbol
Relation	→	Procedure consisting of ground facts (facts without variables)
Tuple	→	Ground fact
Attribute	→	Argument of predicate

- *Example*:

```
person(brown,20,male) <-.
person(jones,21,female) <-.
person(smith,36,male) <-.
```

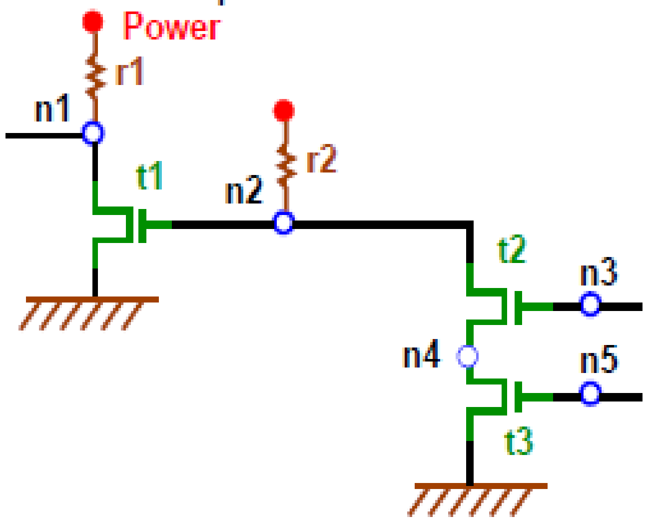
Name	Age	Sex
Brown	20	M
Jones	21	F
Smith	36	M

- *Example*:

```
lived_in(brown,london,15) <-.
lived_in(brown,york,5) <-.
lived_in(jones,paris,21) <-.
lived_in(smith,brussels,15) <-.
lived_in(smith,santander,5) <-.
```

Name	Town	Years
Brown	London	15
Brown	York	5
Jones	Paris	21
Smith	Brussels	15
Smith	Santander	5

- Another example:




```
resistor(power,n1) <-.
resistor(power,n2) <-.

transistor(n2,ground,n1) <-.
transistor(n3,n4,n2) <-.
transistor(n5,ground,n4) <-.
```

```
inverter(Input,Output) <-
   transistor(Input,ground,Output), resistor(power,Output).
nand_gate(Input1,Input2,Output) <-
   transistor(Input1,X,Output), transistor(Input2,ground,X),
   resistor(power,Output).
and_gate(Input1,Input2,Output) <-
   nand_gate(Input1,Input2,X), inverter(X, Output).
```

- Query `and_gate(In1,In2,Out)` has solution: `{In1=n3, In2=n5, Out=n1}`

TREE

C_1: `pet(X) <- animal(X), barks(X).`

C_2: `pet(X) <- animal(X), meows(X).`

C_3: `animal(spot) <-.`

C_4: `animal(barry) <-.`

C_5: `animal(hobbes) <-.`

C_6: `barks(spot) <-.`

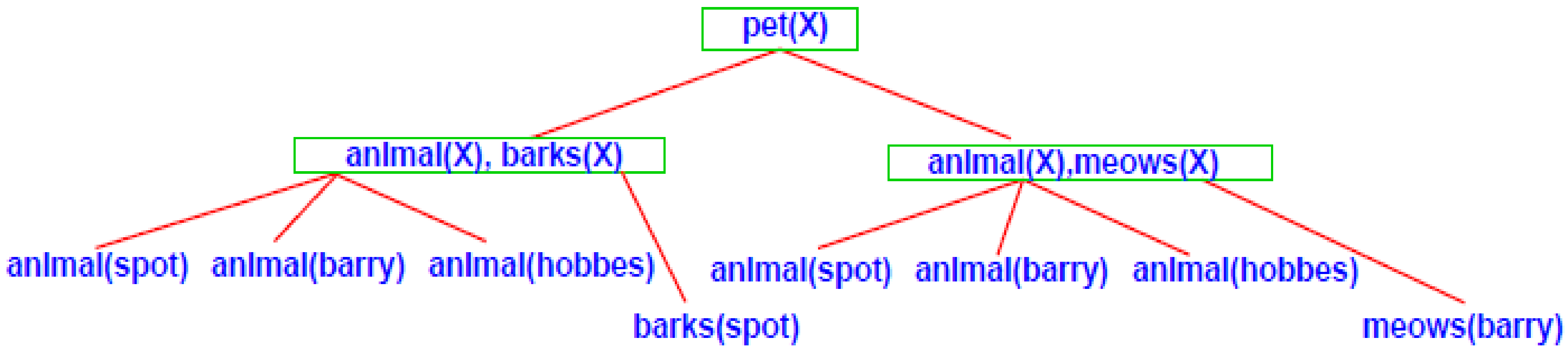
C_7: `meows(barry) <-.`

C_8: `roars(hobbes) <-.`

SEARCH TREE

- A query + a logic program together specify a search tree.

  Example: query ← `pet(X)` with the previous program generates this search tree (the boxes represent the "and" parts [except leaves]):



- The details of the operational semantics explain how the search tree will be explored during execution.
- Different query → different tree.

Graph

graphs are simple and understandable. Graph-theoretic structures are able to represent properties of programs. Known properties of graphs can help improve the understanding of structural properties of the programs logic.

Assume given a set of facts of the form `father(name1,name2)` (`name1` is the father of `name2`).

1. Define a predicate `brother(X,Y)` which holds iff `X` and `Y` are brothers.
2. Define a predicate `cousin(X,Y)` which holds iff `X` and `Y` are cousins.
3. Define a predicate `grandson(X,Y)` which holds iff `X` is a grandson of `Y`.
4. Define a predicate `descendent(X,Y)` which holds iff `X` is a descendent of `Y`.
5. Consider the following genealogical tree:

```
father(a,b).  % 1
father(a,c).  % 2
father(b,d).  % 3
father(b,e).  % 4
father(c,f).  % 5
```

whose graphical representation is:

```
     a
    / \
   b   c
  / \  |
 d   e f
```

Say which answers, and in which order, are generated by your definitions for the queries

```
?- brother(X,Y).
?- cousin(X,Y).
?- grandson(X,Y).
?- descendent(X,Y).
```

```
brother(X,Y) :- father(Z,X), father(Z,Y), not(X=Y).    % 6

cousin(X,Y) :- father(Z,X), father(W,Y), brother(Z,W). % 7

grandson(X,Y) :- father(Z,X), father(Y,Z).             % 8

descendent(X,Y) :- father(Y,X).                        % 9
descendent(X,Y) :- father(Z,X), descendent(Z,Y).       % 10

?- brother(X,Y).
X = b  Y = c ;
X = c  Y = b ;
X = d  Y = e ;
X = e  Y = d ;
No
```

```
?- cousin(X,Y).
X = d  Y = f ;
X = e  Y = f ;
X = f  Y = d ;
X = f  Y = e ;
No

?- grandson(X,Y).
X = d  Y = a ;
X = e  Y = a ;
X = f  Y = a ;
No

?- descendent(X,Y).
X = b  Y = a ;
X = c  Y = a ;
X = d  Y = b ;
X = e  Y = b ;
X = f  Y = c ;
X = d  Y = a ;
X = e  Y = a ;
X = f  Y = a ;
No
```

draw the SLD-tree for the first query.