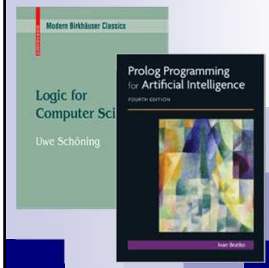


FLORIDA INTERNATIONAL UNIVERSITY

Logic Programming and PROLOG (I)

Dr. Antonio L. Bajuelos

FIU School of Computing & Information Sciences

Note: The most of the information of these slides was extracted and adapted from Schoning's book, "Logic for Computer Scientists". They are provided for COT-3541 students only. Not to be published or publicly distributed without permission by the publisher.

Logic Programming. The Paradigm

- An important programming **paradigm** is to express a program as a **set of rules**.
- The rules are independent and often unordered.
- We'll take a brief look at a particular paradigm - **Logic Programming**.
- And at **PROLOG****, the most successful of the **logic programming languages**.

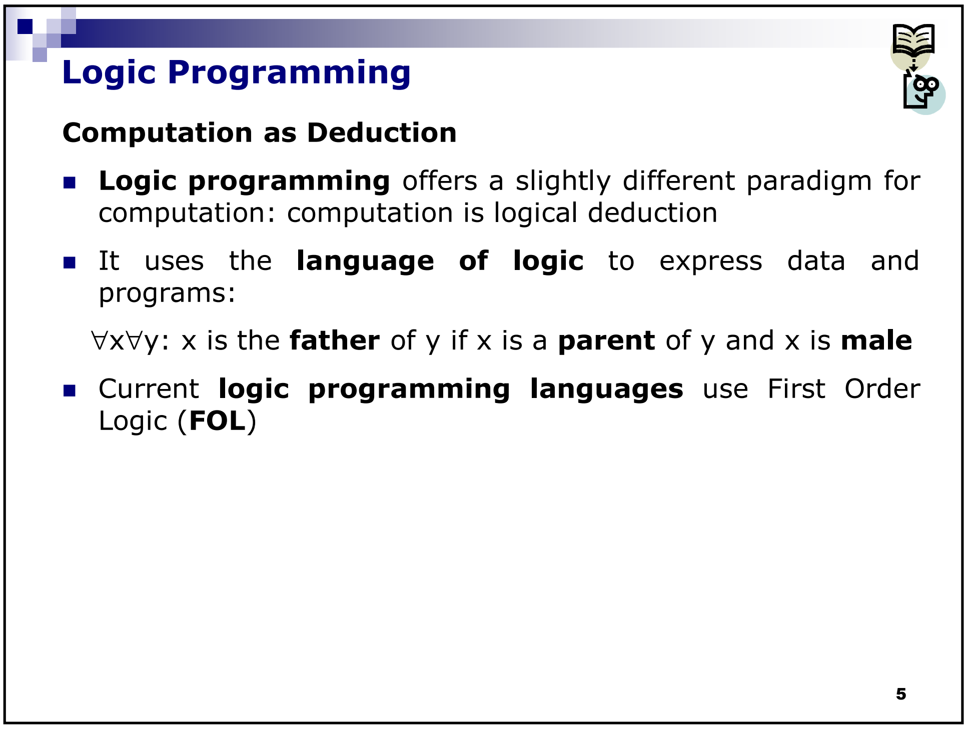
**Recommended text: "PROLOG Programming for Artificial Intelligence", Ivan Bratko, Addison Wesley (3rd or 4th edition)

Logic Programming. The Paradigm

- Some (overlapping) perspectives on **logic programming**:
  - □ Computations as Deduction.
  - □ Theorem Proving.
  - □ Non-procedural Programming.
  - □ Algorithms minus Control.
  - □ A Very High Level Programming Language.
  - □ A Procedural Interpretation of Declarative Specifications.
  - □ …

Logic Programming. History

- **Logic Programming** has roots going back to early **Artificial Intelligence (AI)** researchers like **John McCarthy** in the 50s & 60s
- **Alain Colmerauer** (France) designed **PROLOG** as the first **LP** language <u>in the early 1970s</u>
- **Bob Kowalski** and colleagues in the UK evolved the language to its current form <u>in the late 70s</u>
- It's been widely used for many **AI** systems, but also for systems that need **<u>a fast, efficient and clean rule based engine</u>**
- **PROLOG** is a programming language for symbolic, non-numeric computation
- **PROLOG** is specially well suited for solving problems that involve objects and relations between objects.



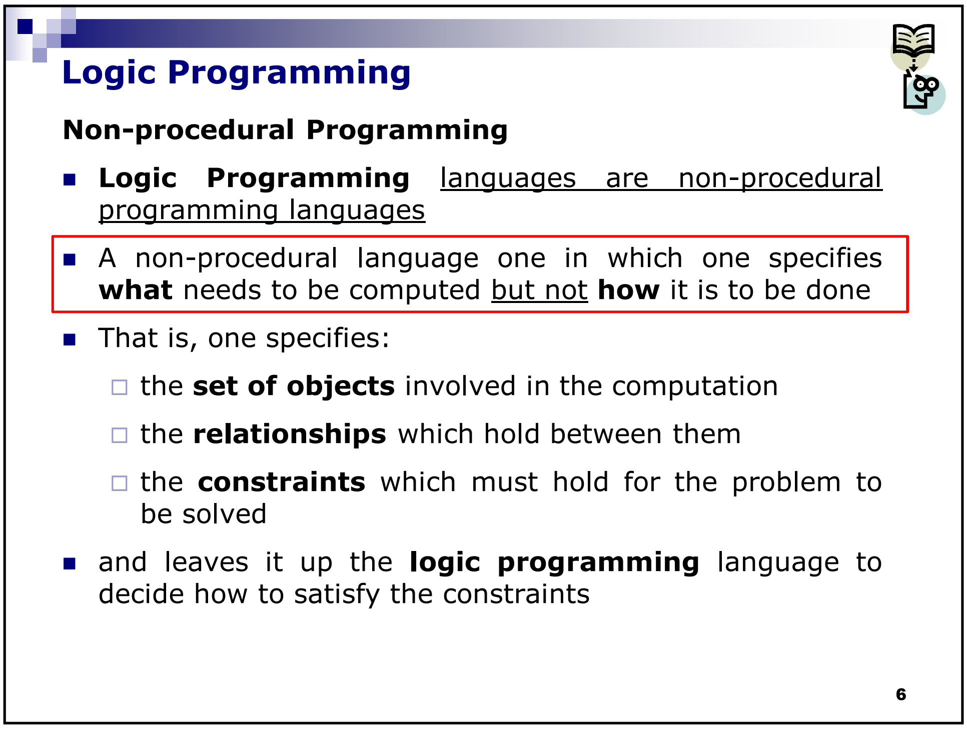
Logic Programming

Computation as Deduction

- **Logic programming** offers a slightly different paradigm for computation: computation is logical deduction
- It uses the **language of logic** to express data and programs:

  ∀x∀y: x is the **father** of y if x is a **parent** of y and x is **male**

- Current **logic programming languages** use First Order Logic (**FOL**)



Logic Programming

Non-procedural Programming

- **Logic Programming** languages are non-procedural programming languages
- A non-procedural language one in which one specifies **what** needs to be computed but not **how** it is to be done
- That is, one specifies:
  - □ the **set of objects** involved in the computation
  - □ the **relationships** which hold between them
  - □ the **constraints** which must hold for the problem to be solved
- and leaves it up the **logic programming** language to decide how to satisfy the constraints

Logic Programming

Procedural vs. Logic Programming

- **Procedural paradigm:** to compute the sum of the element of the list, iterate through the list adding each value to an accumulator variable:

```
int sum(int[] list ) {
   int result = 0;
   for(int i=0; i<list.length; ++i) {
     result += list[i];
   }
   return result;
}
```

- **Logic paradigm**

```
% the sum of the empty list is zero
sum([],0).

% the sum of the list with head H and
% tail T is N if the sum of the list T
% is M and N is M + H
sum([H|T],N) :- sum(T,M), N is M+H.
```

(PRO)gramming en (LOG)ique (PROLOG)

A Simple PROLOG Model

- Think of **PROLOG** as a system which has a "database" composed of **two components**:
  - □ **facts:** statements about true relations which hold between particular objects in the world.
    - **Example:**

parent(bob,john).	% bob is a parent of john
parent(nancy,john).	% nancy is a parent of john
male(bob).	% bob is male

  - □ **rules:** statements about relations between objects in the world which use variables to express generalizations.

% X is the **father** of Y if X is a **parent** of Y and X is **male**

father(X,Y) :- parent(X,Y), male(X).

% X is a **sibling** of Y if X and Y share a **parent**

sibling(X,Y) :- parent(P,X), parent(P,Y).

PROLOG

```
parent(bob,john).        % bob is a parent of john
parent(nancy,john).      % nancy is a parent of john
male(bob).               % bob is male

% X is the father of Y if X is a parent of Y and X is male
father(X,Y) :- parent(X,Y), male(X).
% X is a sibling of Y if X and Y share a parent
sibling(X,Y) :- parent(P,X), parent(P,Y).
```

Queries:

- In **PROLOG** we also have **queries** in addition to having **facts** and **rules.**
- A program in **PROLOG** is partly like a **database** but much more powerful since we can also have general rules to **infer new facts.**
- **PROLOG** interpreter can follow the facts and rules and answer queries by sophisticated **search**.
- A simple **query** is just a predicate that might have variables in it. For example:

  ?- parent(bob, X). % Who are Bob's children?

  ?- parent(X, liz). % Who is Liz's parent?

PROLOG

A simple PROLOG program:

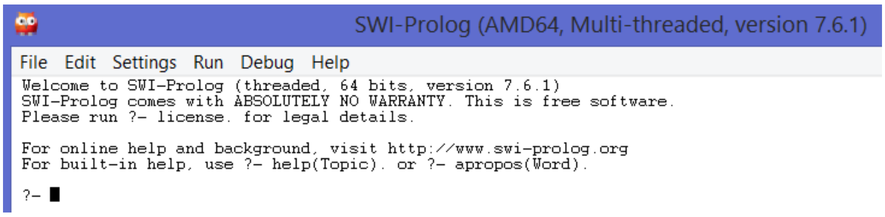
```
male(robert).             %this is our family.pl program
male(john).
female(nancy).
female(victoria).
parent(robert,john).   % robert is a parent of john
parent(victoria,john).
father(X,Y):-parent(X,Y), male(X).
mother(X,Y):-parent(X,Y), female(X).
```

- **Some comments:**
  - □ A **fact/rule** (statement) ends with "**.**" and white space ignored
  - □ read **:-** after rule head as "**if**".
  - □ Read **comma** in body as "**and**"
  - □ **Comment** a line with **%** or use **/* */** for multi-line comments

PROLOG

Running PROLOG

- A commercial version: **sicstus-prolog**
- A good **free version**: **swi-prolog** (**http://www.swi-prolog.org/**)
  - (available for **Linux**, **MacOS X** and **Windows**)
- We will use **swi-prolog** throughout this course

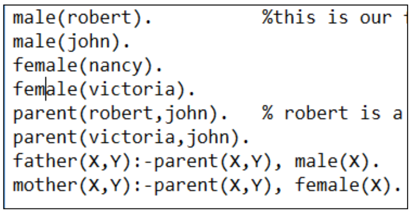

SWI-Prolog (AMD64, Multi-threaded, version 7.6.1)
File Edit Settings Run Debug Help

```
Welcome to SWI-Prolog (threaded, 64 bits, version 7.6.1)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- 
```

- ?- ← the prompt. You can load your program and ask queries
- ?- consult(family1). % loading your program
  % we can use full-name with quotation 'family.pl'
- ?- halt. exit from swi-prolog

PROLOG

Running PROLOG. First example

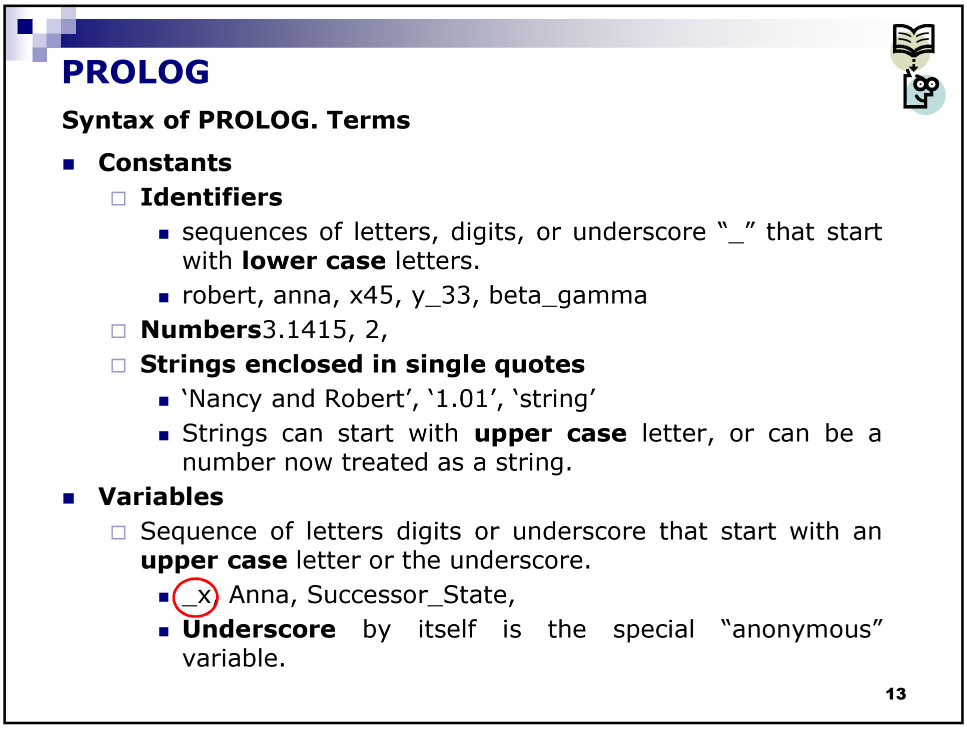


```
male(robert).          %this is our
male(john).
female(nancy).
female(victoria).
parent(robert,john).   % robert is a
parent(victoria,john).
father(X,Y):-parent(X,Y), male(X).
mother(X,Y):-parent(X,Y), female(X).
```

?- male(robert).
true. % the above was true
?- male(victoria).
false. % the above was false
?- male(mycat).
false.

?- male(X). % X is a **variable**, we are asking "**who** is male?"
X=robert; % now type **semicolon** to ask for more answers.
X=john.

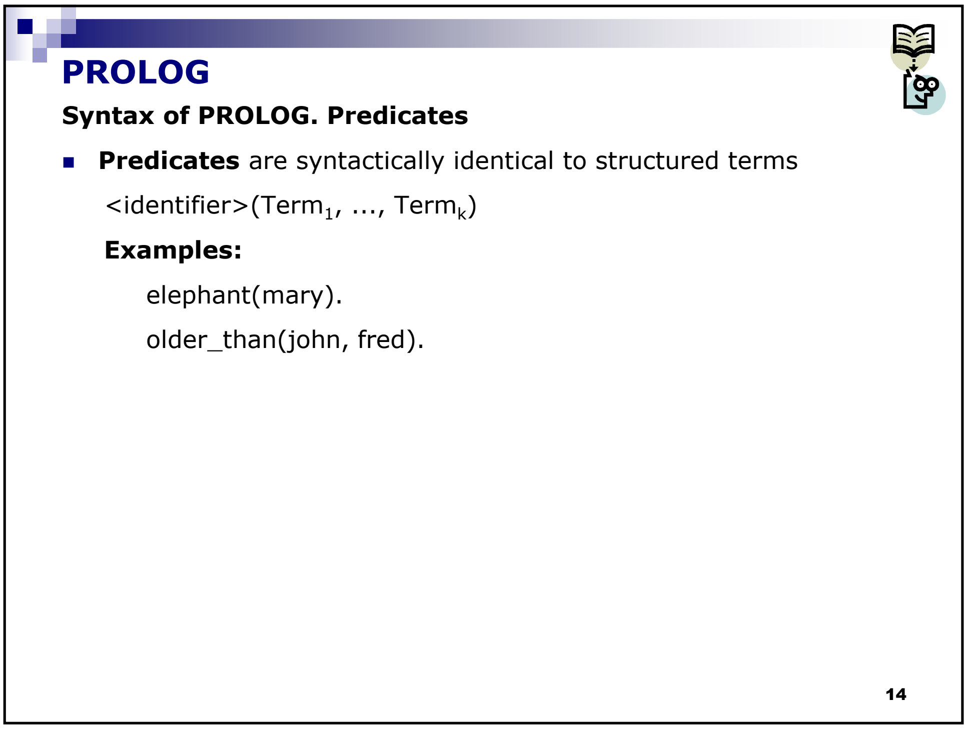
?- father(F,C). %F & C are variables, we are asking "**who** is father of **whom**"
F=robert, C=john;
false.



PROLOG

Syntax of PROLOG. Terms

- **Constants**
  - **Identifiers**
    - sequences of letters, digits, or underscore "_" that start with **lower case** letters.
    - robert, anna, x45, y_33, beta_gamma
  - **Numbers**3.1415, 2,
  - **Strings enclosed in single quotes**
    - 'Nancy and Robert', '1.01', 'string'
    - Strings can start with **upper case** letter, or can be a number now treated as a string.
- **Variables**
  - Sequence of letters digits or underscore that start with an **upper case** letter or the underscore.
    - _x Anna, Successor_State,
    - **Underscore** by itself is the special "anonymous" variable.



PROLOG

Syntax of PROLOG. Predicates

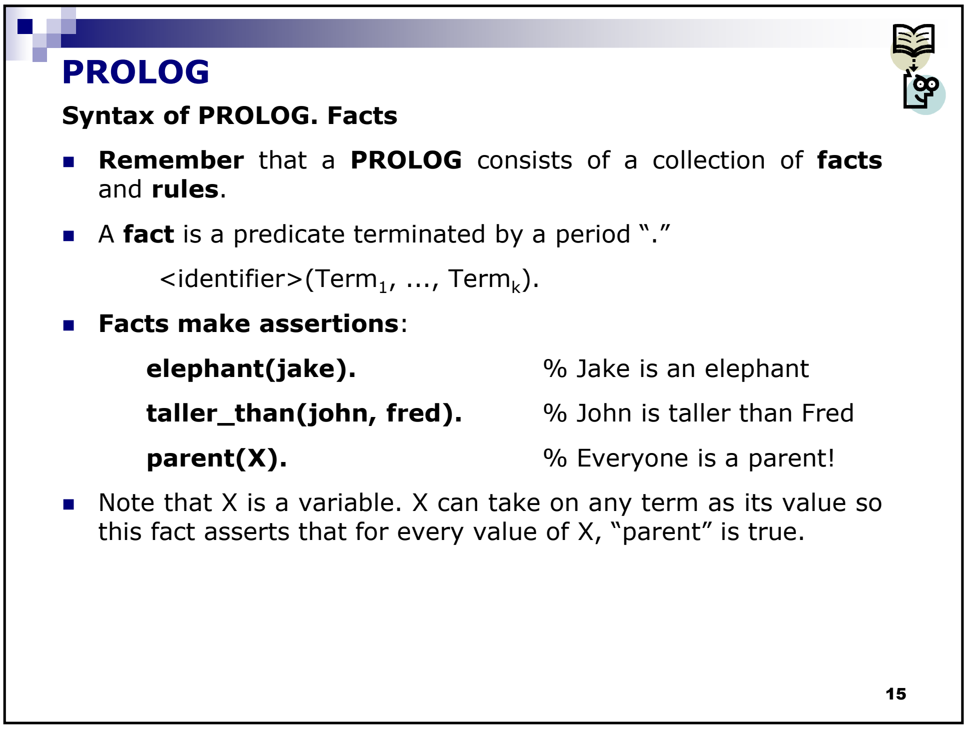
- **Predicates** are syntactically identical to structured terms

  <identifier>(Term$_1$, ..., Term$_k$)

  Examples:

  elephant(mary).

  older_than(john, fred).



PROLOG

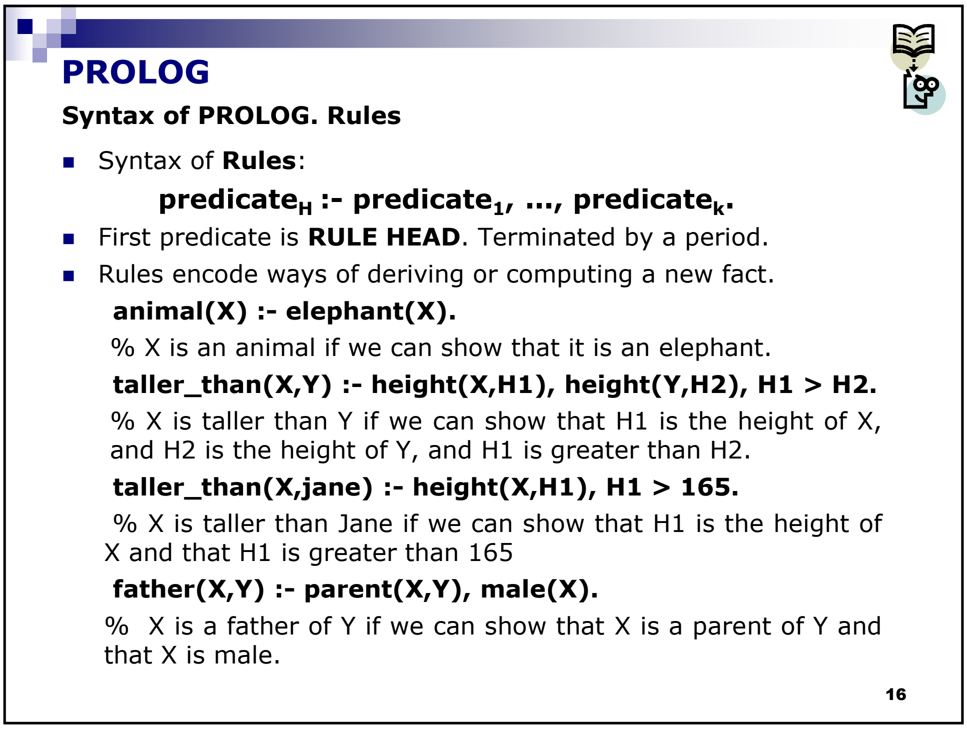
Syntax of PROLOG. Facts

- **Remember** that a **PROLOG** consists of a collection of **facts** and **rules**.
- A **fact** is a predicate terminated by a period "."

  <identifier>(Term_1, …, Term_k).
- **Facts make assertions**:

```
elephant(jake).            % Jake is an elephant
taller_than(john, fred).   % John is taller than Fred
parent(X).                 % Everyone is a parent!
```

- Note that X is a variable. X can take on any term as its value so this fact asserts that for every value of X, "parent" is true.



PROLOG

Syntax of PROLOG. Rules

- Syntax of **Rules**:

  predicate_H :- predicate_1, …, predicate_k.
- First predicate is **RULE HEAD**. Terminated by a period.
- Rules encode ways of deriving or computing a new fact.

animal(X) :- elephant(X).

% X is an animal if we can show that it is an elephant.

taller_than(X,Y) :- height(X,H1), height(Y,H2), H1 > H2.

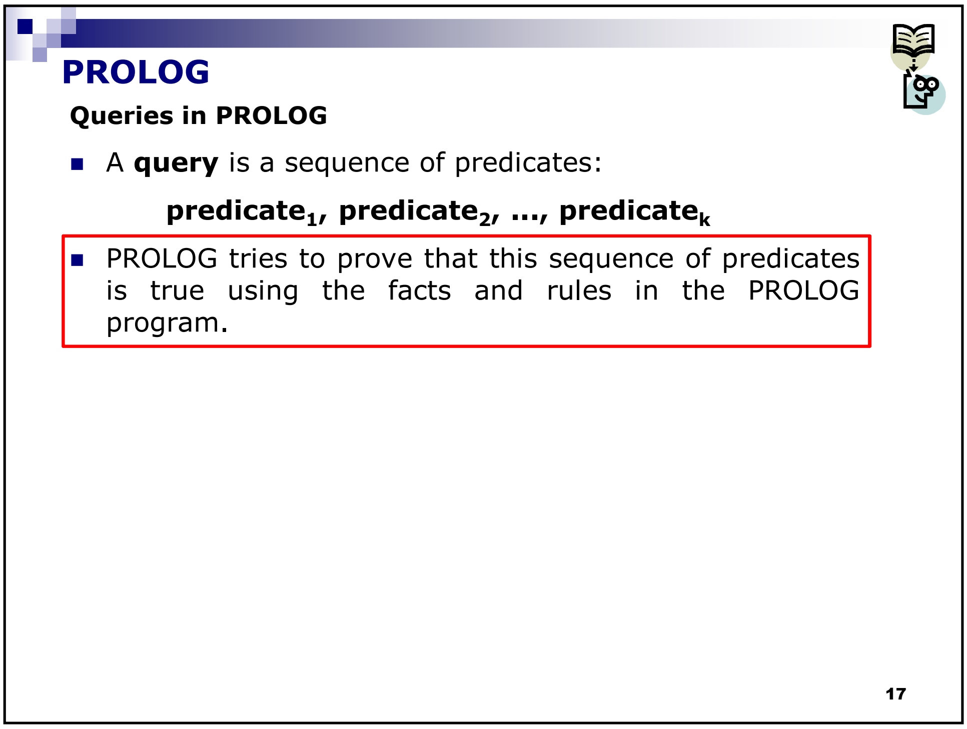
% X is taller than Y if we can show that H1 is the height of X, and H2 is the height of Y, and H1 is greater than H2.

taller_than(X,jane) :- height(X,H1), H1 > 165.

% X is taller than Jane if we can show that H1 is the height of X and that H1 is greater than 165

father(X,Y) :- parent(X,Y), male(X).

% X is a father of Y if we can show that X is a parent of Y and that X is male.



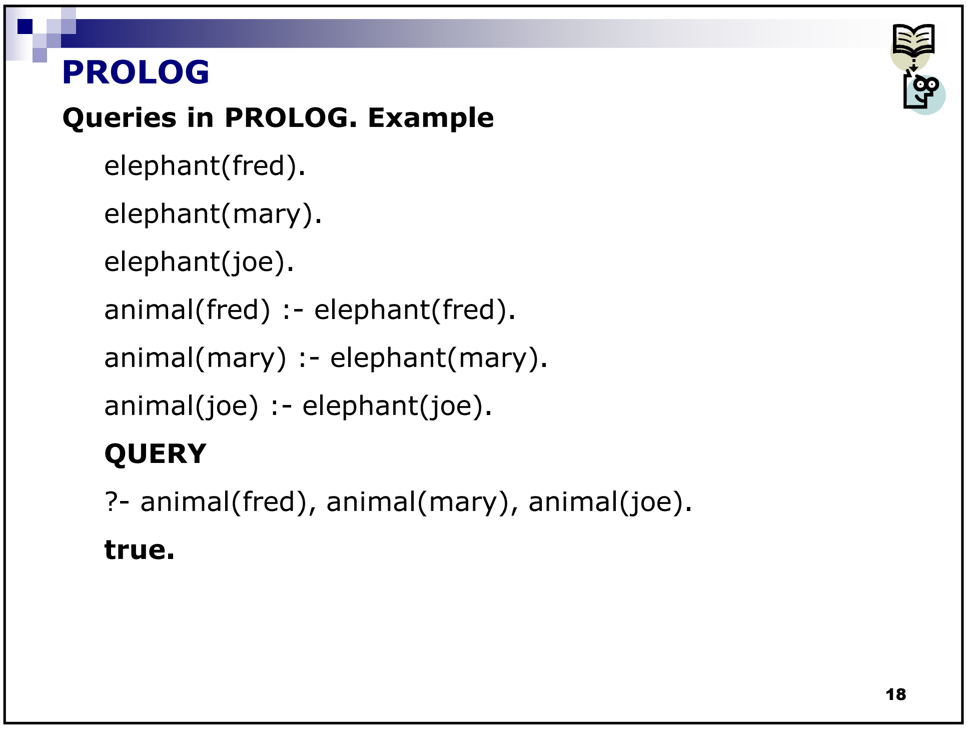
PROLOG

Queries in PROLOG

- A **query** is a sequence of predicates:

  $\mathbf{predicate_1, predicate_2, \ldots, predicate_k}$

- PROLOG tries to prove that this sequence of predicates is true using the facts and rules in the PROLOG program.



PROLOG

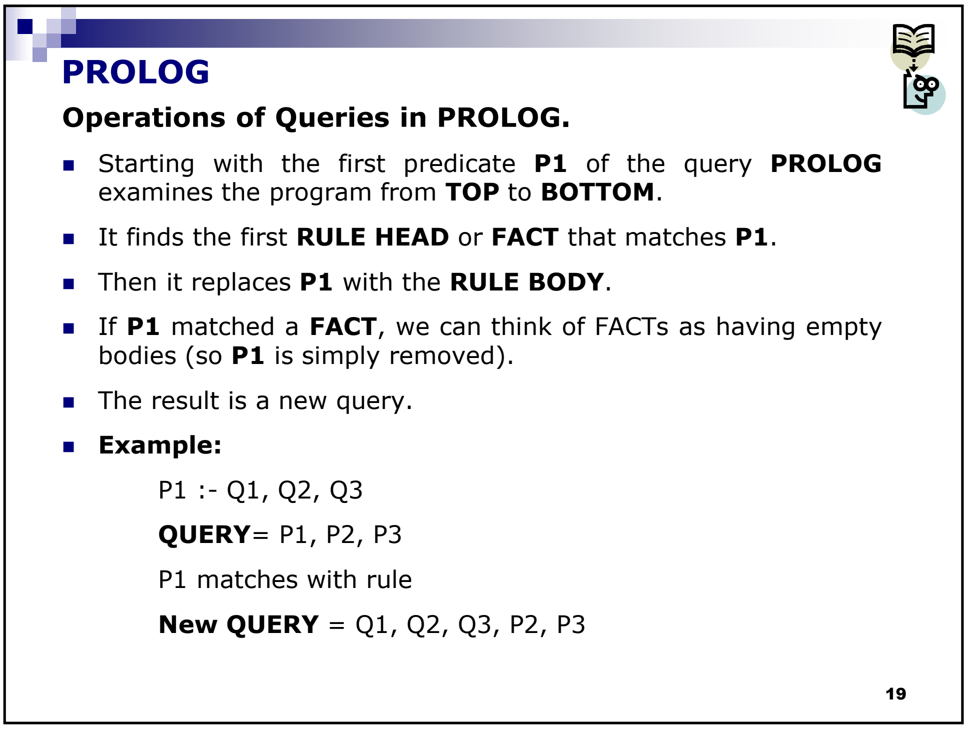
Queries in PROLOG. Example

```
elephant(fred).
elephant(mary).
elephant(joe).
animal(fred) :- elephant(fred).
animal(mary) :- elephant(mary).
animal(joe) :- elephant(joe).
```

QUERY

```
?- animal(fred), animal(mary), animal(joe).
```

true.



PROLOG

Operations of Queries in PROLOG.

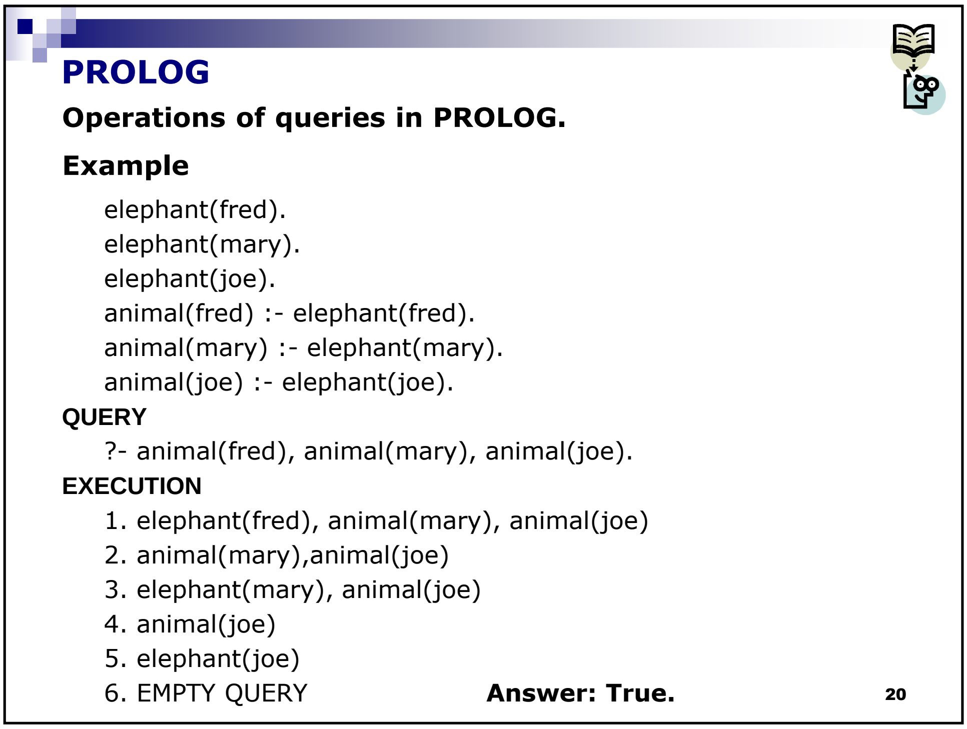
- Starting with the first predicate **P1** of the query **PROLOG** examines the program from **TOP** to **BOTTOM**.
- It finds the first **RULE HEAD** or **FACT** that matches **P1**.
- Then it replaces **P1** with the **RULE BODY**.
- If **P1** matched a **FACT**, we can think of FACTs as having empty bodies (so **P1** is simply removed).
- The result is a new query.
- **Example:**

  P1 :- Q1, Q2, Q3

  QUERY= P1, P2, P3

  P1 matches with rule

  New QUERY = Q1, Q2, Q3, P2, P3



PROLOG

Operations of queries in PROLOG.

Example

```
elephant(fred).
elephant(mary).
elephant(joe).
animal(fred) :- elephant(fred).
animal(mary) :- elephant(mary).
animal(joe) :- elephant(joe).
```

QUERY

```
?- animal(fred), animal(mary), animal(joe).
```

EXECUTION

1. elephant(fred), animal(mary), animal(joe)
2. animal(mary),animal(joe)
3. elephant(mary), animal(joe)
4. animal(joe)
5. elephant(joe)
6. EMPTY QUERY

Answer: True.

PROLOG

Operations of queries in PROLOG.

- **Important points:**
  - □ If this process reduces the query to the **empty query**, PROLOG returns “yes”.
  - □ However, during this process each predicate in the query might match more than one fact or rule head.
  - □ In this case **PROLOG** always choose the **first match** it finds.
  - □ Then if the resulting query reduction did not succeed (i.e., we hit a predicate in the query that does not match any rule head of fact), **PROLOG backtracks** and tries a new match.

PROLOG

Operations of queries in PROLOG.

- **Important points:**
  - □ … **PROLOG** always choose the **first match** it finds. Then if the resulting query reduction did not succeed (i.e., we hit a predicate in the query that does not match any rule head of fact), **PROLOG backtracks** and tries a new match.
- **Example:**

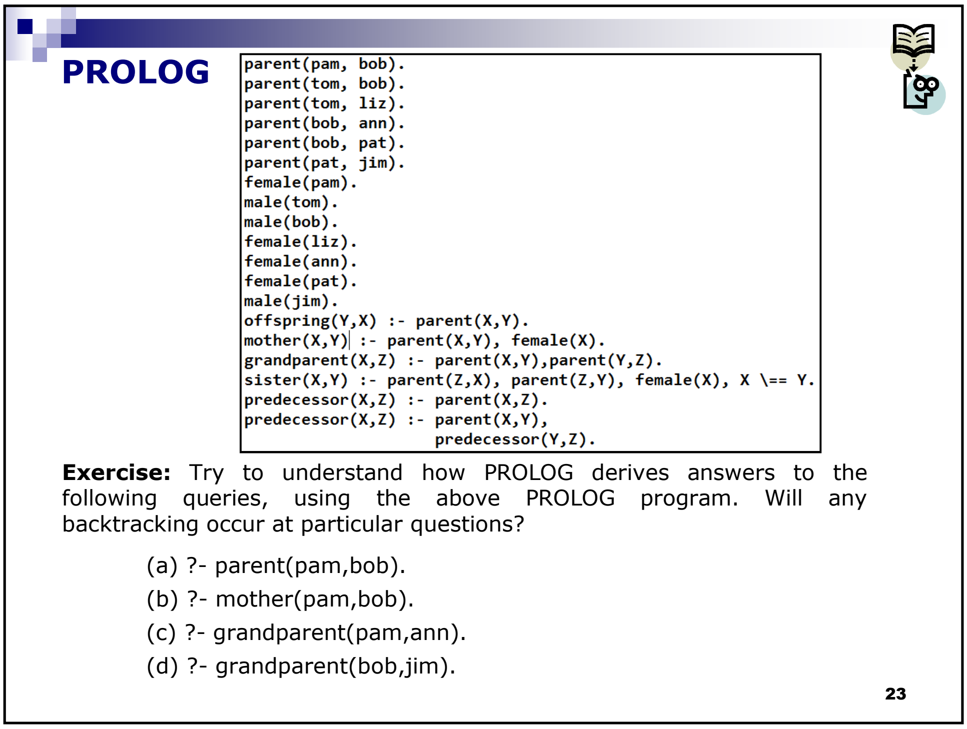
```
ant_eater(fred).
animal(fred) :- elephant(fred).
animal(fred) :- ant_eater(fred).
```

QUERY

```
?- animal(fred).
```

EXECUTION

1. elephant(fred).
2. FAIL, BACKTRACK
3. ant_eater(fred).
4. EMPTY QUERY **Answer: True.**



PROLOG

```
parent(pam, bob).
parent(tom, bob).
parent(tom, liz).
parent(bob, ann).
parent(bob, pat).
parent(pat, jim).
female(pam).
male(tom).
male(bob).
female(liz).
female(ann).
female(pat).
male(jim).
offspring(Y,X) :- parent(X,Y).
mother(X,Y) :- parent(X,Y), female(X).
grandparent(X,Z) :- parent(X,Y),parent(Y,Z).
sister(X,Y) :- parent(Z,X), parent(Z,Y), female(X), X \== Y.
predecessor(X,Z) :- parent(X,Z).
predecessor(X,Z) :- parent(X,Y),
                    predecessor(Y,Z).
```

Exercise: Try to understand how PROLOG derives answers to the following queries, using the above PROLOG program. Will any backtracking occur at particular questions?

(a) ?- parent(pam,bob).

(b) ?- mother(pam,bob).

(c) ?- grandparent(pam,ann).

(d) ?- grandparent(bob,jim).