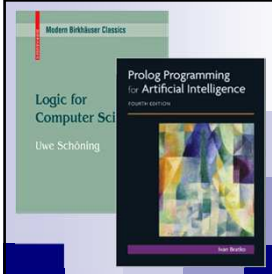


FIU FLORIDA INTERNATIONAL UNIVERSITY

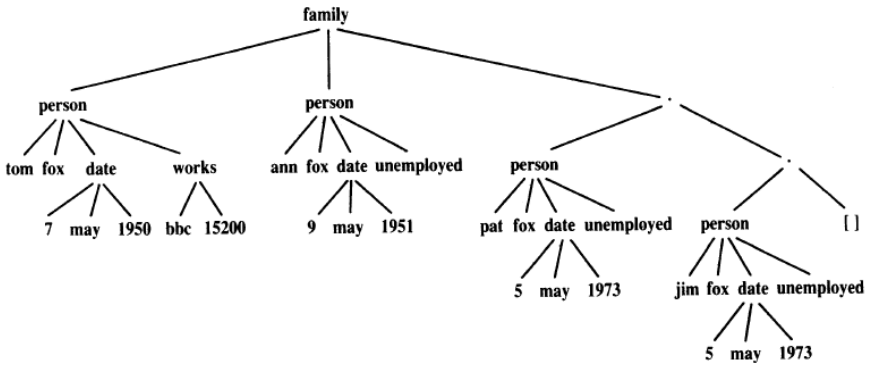
Logic Programming and PROLOG (V)

Dr. Antonio L. Bajuelos

FIU School of Computing & Information Sciences

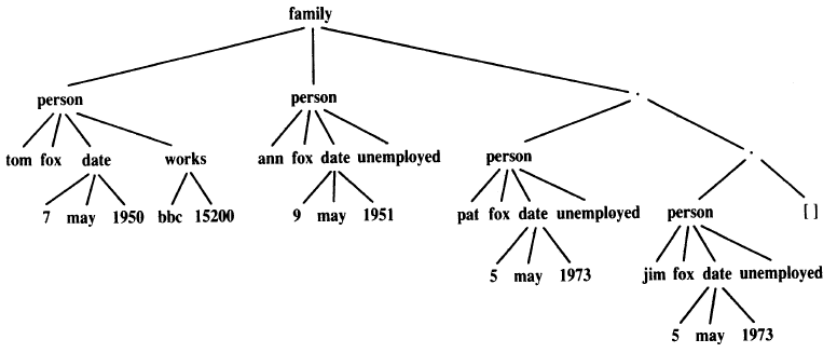
PROLOG. Applications

- **PROLOG** as a **natural database query language**:
- A **database** can be naturally represented in **PROLOG** as a **set of facts**.
- **Example:**
  - A database about families can be represented so that each family is described by one clause

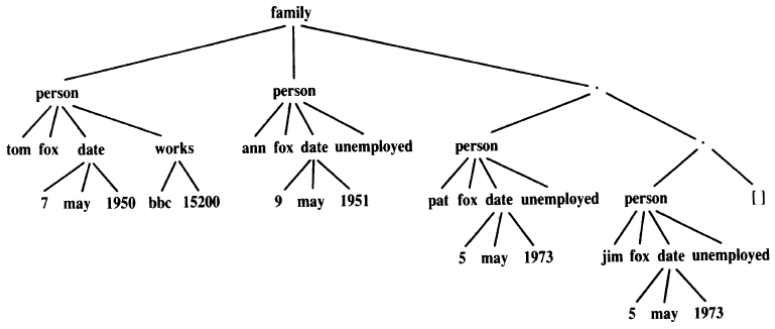


PROLOG. Applications

- **Example:**
  - Each family has **three** components: **husband**, **wife** and **children**.
  - As the number of children varies from family to family the children are represented by a list that is capable of accommodating any number of items.
  - Each person is, in turn, represented by a structure of four components: **name**, **surname**, **date of birth**, **job**. The job information is "unemployed" or it specifies the working organization and salary



PROLOG. Applications



- The "**family**" can be stored in the database by the clause:

```
family(
  person(tom, fox, date(7,may,1950), works(bbc,15200)),
  person(ann, fox, date(9,may, 1951), unemployed),
  [person(pat, fox, date(5,may,1973), unemployed),
   person(jim, fox, date(5,may,1973), unemployed)]).
```

4

PROLOG. Applications

```
family(
  person(tom, fox, date(7,may,1950), works(bbc,15200)),
  person(ann, fox, date(9,may, 1951), unemployed),
  [person(pat, fox, date(5,may,1973), unemployed),
   person(jim, fox, date(5,may,1973), unemployed)]).
```

- In **PROLOG** we can refer to objects without actually specifying all the components of these objects.
- We can merely indicate the *structure* of objects that we are interested in, and leave the particular components in the structures unspecified or only partially specified.
- **Example:** We can refer to all **Fox families** by:

  ?- family(person(_, fox, _, _), _, _).

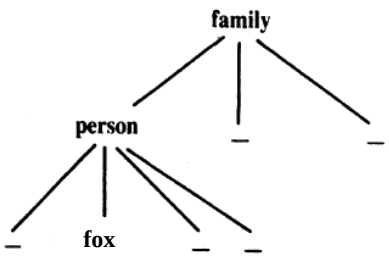
5

PROLOG. Applications

```
family(
  person(tom, fox, date(7,may,1950), works(bbc,15200)),
  person(ann, fox, date(9,may, 1951), unemployed),
  [person(pat, fox, date(5,may,1973), unemployed),
   person(jim, fox, date(5,may,1973), unemployed)]).
```

- **Example:** We can ask if there are **Fox families** by:

  ?- family(person(_, fox, _, _), _, _).



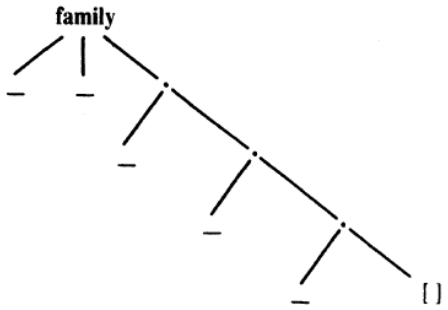
6

PROLOG. Applications

```
family(
  person(tom, fox, date(7,may,1950), works(bbc,15200)),
  person(ann, fox, date(9,may, 1951), unemployed),
  [person(pat, fox, date(5,may,1973), unemployed),
   person(jim, fox, date(5,may,1973), unemployed)]).
```

- **Example:** We can ask if there are families with **three children** by

  ?- family(_, _, [_, _, _]).



PROLOG. Applications

```
family(
  person(tom, fox, date(7,may,1950), works(bbc,15200)),
  person(ann, fox, date(9,may, 1951), unemployed),
  [person(pat, fox, date(5,may,1973), unemployed),
   person(jim, fox, date(5,may,1973), unemployed)]).
```

- **Example:** To find all married women that have at least three children we can pose the question:

  ?- family(_, person(Name, Surname, _, _), [_, _, _ | _]).

PROLOG. Applications

```
family(
  person(tom, fox, date(7,may,1950), works(bbc,15200)),
  person(ann, fox, date(9,may, 1951), unemployed),
  [person(pat, fox, date(5,may,1973), unemployed),
   person(jim, fox, date(5,may,1973), unemployed)]).
```

- We can provide a set of procedures that can serve as a utility to make the interaction with the database more comfortable.
- Some useful utility procedures for our database are:

```
husband(X) :- family(X, _, _).          % X is a husband
wife(X) :- family(_,X, _).              % X is a wife
child(X) :- family( _, _, Children),    % X is a child
            member(X, Children).
member(X, [X|L]).
member(X, [Y|L]) :- member (X,L).
...
```

9

PROLOG. Applications

```
family(
  person(tom, fox, date(7,may,1950), works(bbc,15200)),
  person(ann, fox, date(9,may, 1951), unemployed),
  [person(pat, fox, date(5,may,1973), unemployed),
   person(jim, fox, date(5,may,1973), unemployed)]).
```

- We can provide a set of procedures that can serve as a utility to make the interaction with the database more comfortable.
- Some useful utility procedures for our database are:

```
...
exist(Person) :- husband(Person);      % any person in DB
wife(Person);
child(Person).

dateofbirth (person(_, _, Date, _), Date).
salary(person(_,_, _, works(_, S)),S). % Salary of work. per.
salary(person(_,_,_,unemployed),0).   % Salary of unempl.
```

10

PROLOG. Applications

```
husband(X) :- family(X, _, _).
wife(X) :- family(_, X, _).
child(X) :- family( _, _, Children),
        member(X, Children).

member(X, [X|L]).
member(X, [Y|L]) :- member (X,L).

exist(Person) : - husband(Person);
        wife(Person);
        child(Person).

dateofbirth(person(_, _, Date, _),Date).

salary(person(_,_, _, works(_, S)),S).
salary(person(_,_,_, unemployed),0).
```

- We can use these utilities, for example, in the following queries to the database:
  - Find the **names of all the people** in the database:

    ?- exist(person(Name, Surname, _, _)).

11

PROLOG. Applications

```
husband(X) :- family(X, _, _).
wife(X) :- family(_, X, _).
child(X) :- family( _, _, Children),
        member(X, Children).

member(X, [X|L]).
member(X, [Y|L]) :- member (X,L).

exist(Person) : - husband(Person);
        wife(Person);
        child(Person).

dateofbirth(person(_, _, Date, _),Date).

salary(person(_,_, _, works(_, S)),S).
salary(person(_,_,_, unemployed),0).
```

- We can use these utilities, for example, in the following queries to the database:
  - Find **all children born in 2008**:

    ?- child(X), dateofbirth(X, date(_,_, 2008)).

12

PROLOG. Applications

```
husband(X) :- family(X, _, _).
wife(X) :- family(_, X, _).
child(X) :- family( _, _, Children),
        member(X, Children).

member(X, [X|L]).
member(X, [Y|L]) :- member (X,L).

exist(Person) : - husband(Person);
        wife(Person);
        child(Person).

dateofbirth(person(_, _, Date, _),Date).

salary(person(_,_, _, works(_, S)),S).
salary(person(_,_,_, unemployed),0).
```

- We can use these utilities, for example, in the following queries to the database:
  - Find **all employed wives**:

    ?- wife(person(Name, Surname, _, works(_, _))).

13

PROLOG. Applications

```
husband(X) :- family(X, _, _).
wife(X) :- family(_, X, _).
child(X) :- family( _, _, Children),
        member(X, Children).

member(X, [X|L]).
member(X, [Y|L]) :- member (X,L).

exist(Person) : - husband(Person);
        wife(Person);
        child(Person).

dateofbirth(person(_, _, Date, _),Date).

salary(person(_,_, _, works(_, S)),S).
salary(person(_,_,_, unemployed),0).
```

- We can use these utilities, for example, in the following queries to the database:
  - Find **the names of unemployed people who were born before 1980**:

    ?- exist(person(Name, Surname, date(_, _,Year), unemployed)), Year < 1980.

14

PROLOG. Applications

```
husband(X) :- family(X, _, _).
wife(X) :- family(_, X, _).
child(X) :- family( _, _, Children),
        member(X, Children).

member(X, [X|L]).
member(X, [Y|L]) :- member (X,L).

exist(Person) : - husband(Person);
        wife(Person);
        child(Person).

dateofbirth(person(_, _, Date, _),Date).

salary(person(_,_, _, works(_, S)),S).
salary(person(_,_,_, unemployed),0).
```

- We can use these utilities, for example, in the following queries to the database:
  - Find **people born before 1970 whose salary < 8000**:

    ?- exist(Person),
    dateofbirth(Person, date(_, _, Year)),
    Year < 1970, salary(Person, Salary), Salary < 8000.

15

PROLOG. Applications

```
husband(X) :- family(X, _, _).
wife(X) :- family(_, X, _).
child(X) :- family( _, _, Children),
        member(X, Children).

member(X, [X|L]).
member(X, [Y|L]) :- member (X,L).

exist(Person) : - husband(Person);
        wife(Person);
        child(Person).

dateofbirth(person(_, _, Date, _),Date).

salary(person(_,_, _, works(_, S)),S).
salary(person(_,_,_, unemployed),0).
```

- We can use these utilities, for example, in the following queries to the database:
  - Find the names of families with at least three children:

    ?- family(person(_, Name, _, _), _, [_, _, _ | _]).

16

PROLOG. Applications

```
family(
  person(tom, fox, date(7,may,1950), works(bbc,15200)),
  person(ann, fox, date(9,may, 1951), unemployed),
  [person(pat, fox, date(5,may,1973), unemployed),
   person(jim, fox, date(5,may,1973), unemployed)]).
```

- To calculate the **total income of a family** it is useful to define the sum of salaries of a list of people as a two-argument relation:

  total(List_of_people, Sum_of_their_salaries)

- This relation can be programmed as:

```
total([],0).              % Empty list of people

total([Person | List], Sum) :-
salary(Person, S),        % S - salary of first person
total(List, Rest),        % Rest - sum of salaries of others
Sum is S + Rest.
```

17

PROLOG. Applications

```
family(
  person(tom, fox, date(7,may,1950), works(bbc,15200)),
  person(ann, fox, date(9,may, 1951), unemployed),
  [person(pat, fox, date(5,may,1973), unemployed),
   person(jim, fox, date(5,may,1973), unemployed)]).

total([],0).              % Empty list of people

total( [Person | List], Sum) :-
salary(Person, S),        % S - salary of first person
total(List, Rest),        % Rest - sum of salaries of others
Sum is S + Rest.
```

- The total income of families can then be found by the question:

```
?- family(Husband, Wife, Children),
   total([Husband, Wife | Children], Income).
```

18