FIU FLORIDA INTERNATIONAL UNIVERSITY

Logic Programming and PROLOG (II)

Dr. Antonio L. Bajuelos

FIU School of Computing & Information Sciences

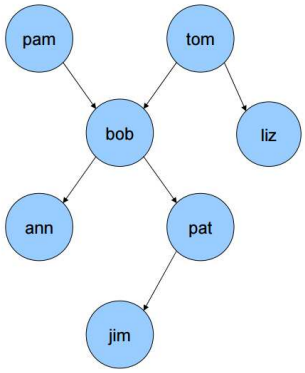
PROLOG

- How **PROLOG** answers questions?
  - **PROLOG** accepts **facts** and **rules** as a **set of axioms**, and the user's question as a conjectured theorem; then it tries to prove this theorem - that is, to demonstrate that it can be logically derived from the axioms.
- **Example:**
  - Let the **axioms** be:
    - All men are fallible.
    - Socrates is a man.
  - A **theorem** that logically follows from these two axioms is:
    - Socrates is fallible.
  - **PROLOG** version

    fallible(X) :- man(X).
    man(socrates).
    ?- fallible(socrates).
    true.

2

PROLOG. Family Tree



Who is a grandparent of jim?

?- parent(Y, jim), parent(X, Y).

X = bob,

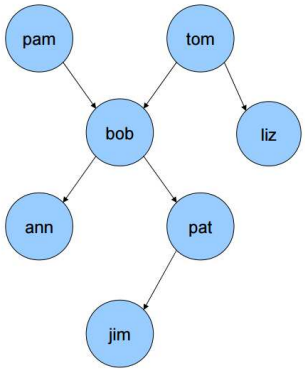
Y = pat ;

false.

```
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).
different(X, Y) :- X \== Y.
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),
               different(X,Y).
r1 predecessor(X,Z) :- parent(X,Z).
r2 predecessor(X,Z) :- parent(X,Y),
                       predecessor(Y,Z).
```

3

PROLOG. Family Tree



?- predecessor(pam, X).

X = bob;

X = ann;

X = pat;

X = jim;

false.

```
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).
different(X, Y) :- X \== Y.
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),
               different(X,Y).
r1 predecessor(X,Z) :- parent(X,Z).
r2 predecessor(X,Z) :- parent(X,Y),
                       predecessor(Y,Z).
```

4

PROLOG

Answers questions? (cont…)

- **Example #2**
  ?- **predecessor(tom, pat).**
- We know that
  parent(bob, pat) is a fact
- Using this fact and a rule r1 we can conclude
  predecessor(bob,pat). This is a derived fact.
- Using this derived fact and the fact
  parent(tom,bob)
- We can conclude using the rule r2 that
  predecessor(tom,pat) is **true**.
- This whole **inference process** of two steps can be written as:
  parent(bob,pat) ⇨ predecessor(bob, pat)
  parent(tom, bob) and predecessor(bob,pat) ⇨ predecessor(tom, pat)

```
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).
different(X, Y) :- X \== Y.
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),
               different(X,Y).
r1 predecessor(X,Z) :- parent(X,Z).
r2 predecessor(X,Z) :- parent(X,Y),
                       predecessor(Y,Z).
```

5

PROLOG

Example #2
?- **predecessor(tom, pat).**

How Prolog find the answers?

- Prolog starts with the goals and, using rules, substitutes the current goals with new goals, until new goals happen to be simple facts.
- Prolog first tries that clause which appears first in the program (r1):
  predecessor(X,Z) :- parent(X, Z).
- Since the goal is
  predecessor(tom, pat)
- The variables in the rule must be instantiated as follows:
  X=tom, Z=pat
- The original goal
  predecessor(tom, pat)
  is then replaced by a new goal:
  parent(tom, pat)

```
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).
different(X, Y) :- X \== Y.
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),
               different(X,Y).
r1 predecessor(X,Z) :- parent(X,Z).
r2 predecessor(X,Z) :- parent(X,Y),
                       predecessor(Y,Z).
```

6

PROLOG

Example #2

?- **predecessor(tom, pat).**

How PROLOG find the answers? **(cont…)**

- …

  parent(tom, pat)

- There is no clause in the program whose head matches the goal parent(tom, pat), therefore this goal fails.
- Now PROLOG **backtracks** to the original goal in order to try an alternative way to derive the top goal **predecessor(tom,pat)**.
- PROLOG try the rule r2.
- As before, the variables X and Z become instantiated as:

  X= tom and Z=pat

- But Y is not instantiated yet. The top goal predecessor(tom,pat) is replaced by two goals:

  parent(tom,Y),

  predecessor(Y,pat)

```
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).
different(X, Y) :- X \== Y.
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),
               different(X,Y).
r1 predecessor(X,Z) :- parent(X,Z).
r2 predecessor(X,Z) :- parent(X,Y),
                       predecessor(Y,Z).
```

7

PROLOG

Example #2

?- **predecessor(tom, pat).**

How PROLOG find the answers? **(cont…)**

- …

  parent(tom,Y) and

  predecessor(Y ,pat)

- PROLOG tries to satisfy them in the order that they are written. The first one is easy as it matches one of the facts in the program. The matching forces **Y** to become instantiated to **bob**.
- Thus the first goal has been satisfied, and the remaining goal has become:

  predecessor(bob, pat)

- To satisfy this goal the rule r1 is used again. Therefore, PROLOG uses a new set of variables in the rule each time the rule is applied. To indicate this we shalt rename the variables in rule r1 for this application as follows:

  predecessor(X', Z') :- parent(X',Z').

```
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).
different(X, Y) :- X \== Y.
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),
               different(X,Y).
r1 predecessor(X,Z) :- parent(X,Z).
r2 predecessor(X,Z) :- parent(X,Y),
                       predecessor(Y,Z).
```

8

PROLOG

Example #2

?- **predecessor(tom, pat).**

How PROLOG find the answers? **(cont…)**

- …

  predecessor(X', Z') :-

  parent(X',Z').

- The head has to match our current goal **predecessor(bob,pat)**.
- Therefore

  X' = bob, Z' = pat
- The current goal is replaced by

  parent(bob,pat)
- This goal is immediately satisfied because it appears in the program as a fact.
- When prolog discovers that a branch fails it automatically **backtracks** to the previous node and tries to apply an alternative clause at that node.

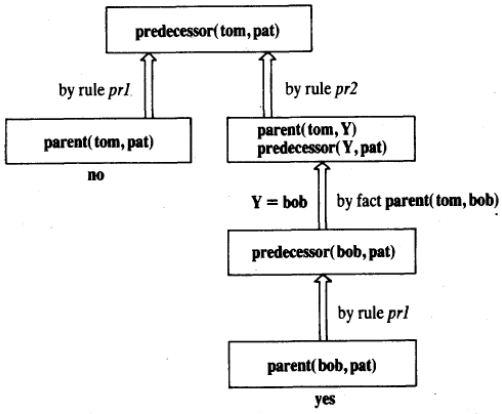
```
parent(pam, bob).
parent(tom, bob).
parent(tom, liz).
parent(bob, ann).
parent(bob, pat).
parent(pat, jim).
female(pam).
male(tom).
male(bob).
female(liz).
female(ann).
female(pat).
male(jim).
different(X, Y) :- X \== Y.
offspring(Y,X) :- parent(X,Y).
mother(X,Y) :- parent(X,Y),
               female(X).
grandparent(X,Z) :- parent(X,Y),
                    parent(Y,Z).
sister(X,Y) :- parent(Z,X),
               parent(Z,Y),
               female(X),
               different(X,Y).
r1 predecessor(X,Z) :- parent(X,Z).
r2 predecessor(X,Z) :- parent(X,Y),
                       predecessor(Y,Z).
```

9

PROLOG

Example #2

?- **predecessor(tom, pat).**




```
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).
different(X, Y) :- X \== Y.
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),
               different(X,Y).
pr1 predecessor(X,Z) :- parent(X,Z).
pr2 predecessor(X,Z) :- parent(X,Y),
                        predecessor(Y,Z).
```

10

PROLOG

The first PROLOG summary.

- **PROLOG** programming consists of defining relations and querying about relations.
- A program consists of clauses. These are of three types: **facts**, **rules** and **queries** (or questions).
- A relation can be specified by facts, simply stating the n-tuples of objects that satisfy the relation, or by stating rules about the relation.
- **PROLOG** answer to a question consists of a set of objects that satisfy the question.
- In **PROLOG**, to establish whether an object satisfies a query is often a complicated process that involves **logical inference**, exploring among alternatives and possibly **backtracking**.
- The inference process is done automatically by the **PROLOG** system and is, in principle, hidden from the user.

11

PROLOG

The use of the "_" in PROLOG

- When a variable appears in a clause once only, we do not have to invent a name for it.
- We can use the so-called "**anonymous**" variable, which is written as a single **underscore character**.
- **Example:**

  hasachild(X) :- parent(X,Y).

  % X has a child if X is a parent of some Y.
- We can defining the property **hasachild** which,

  hasachild(X) :- parent(X, _).

12

PROLOG

The use of the "_" in PROLOG (cont...)

- We can say that there is somebody who has a child if there are two objects such that one is a parent of the other:

  somebody_has_child :- parent(_, _).

- This is equivalent to:

  somebody_has_child :- parent(X,Y).

- But this is, of course, quite different from:

  somebody_has_child :- parent(X,X).

- If the anonymous variable appears in a question clause then its value is not output when PROLOG answers the question.
- If we are interested in people who have children, but not in the names of the children, then we can simply ask:

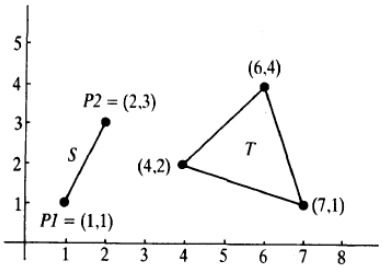
  ?- parent(X,_).

13

PROLOG

Structures

- Structured objects (or simply **structures**) are objects that have several components.
- The objects in the figure be represented by the following **PROLOG** terms:



  P1 = point(1,1)
  P2 = point(2,3)
  S = seg(P1,P2) = seg(point(1,1), point(2,3))
  T = triangle(point(4,2), point(6,4), point(7,1))

- **point**, **seg** and **triangle** are the **functors** of these structures

14

PROLOG

Matching

- The most important operation on terms is **matching**.
- Given two terms, we say that they **match** if:
  - they are identical, or
  - the variables in both terms can be instantiated to objects in such a way that after the substitution of variables by these objects the terms become identical.
- **Example:**
  - Terms **date(D,M,1830)** and **date(D1,may,Y1)** match.
  - One instantiation that makes both terms identical is:
    - D is instantiated to D1 (D=D1)
    - M is instantiated to may (M=may)
    - Y1 is instantiated to 1830 (Y1=1830)
- Note that terms **date(D,M,1830)** and **date(D1,M1,1444)** do **not match**, nor do the terms date(X,Y,Z) and point(X,Y,Z).

15

PROLOG

Matching (cont...)

- Matching is a process that takes as input two terms and checks whether they match. If the terms do not match we say that this process fails.
- If they do match then the process succeeds and it also instantiates the variables in both terms to such values that the terms become identical.
- **Example:**

```
?- date(D,M,1830) = date(D1,may,Y1).
    D = D1,
    M = may,
    Y1 = 1830.

?- date(D,M,1830) = date(D1,may,1492).
false
```

16

PROLOG

Matching (cont...)

- Matching in **PROLOG** always results in the **most general** instantiation.
- Example

  ?- **date(D, M,1830) = date(D1, may,Y1),**
  date(D,M,1830)= date(15,M,Y).

  - To satisfy the **first goal**, PROLOG instantiates the variables as follows:

    D=D1
    M=may
    Y1=1830

  - After having satisfied the **second goal**, the instantiation be comes more specific as follows:

    D=15
    D1=15
    M= may
    Y1=1830
    Y=1830

17

PROLOG

Matching (cont...)

- The general rules to decide whether two terms, S and T, match are as follows:
  - If S and T are constants then S and T match only if they are the same object.
  - Is S is a variable and T is anything, then they match, and S is instantiated to T (substitution [S/T]).
  - If S and T are structures then they match only if:
    - S and T have the same principal **functor**, and
    - All their corresponding components match.

18

PROLOG

Monkey and Banana

- The problem:
  - There is a **monkey** at the door into a room.
  - In the middle of the room a **banana** is hanging 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.

Can the monkey get the banana?

19

PROLOG

Monkey and Banana (cont...)

- The **monkey** can perform the following **actions**:
  - **Walk** on the floor
  - **Climb** the box
  - **Push** the box around (if it is already at the box)
  - **Grasp** the **banana** if standing on the box directly under the banana.

20

PROLOG

Monkey and Banana (cont…)

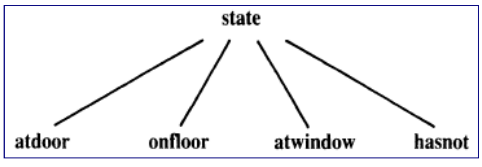
- **Monkey World** is described by some **state** that can change in time.
- **Current state** is determined by the position of the objects.
- **States:**
  - **Monkey location** (in the room)
  - **Monkey position**: Horizontal or Vertical
  - **Box location** (in the room)
  - **Monkey has the banana?** (Yes or Not)

21

PROLOG

Monkey and Banana (cont…)

- **Initial State:**
  - Monkey is at the door
  - Monkey is on floor
  - Box is at window
  - Monkey does not have banana



- In **PROLOG**

  state(atdoor, onfloor, atwindow, hasnot).

- The **goal of the game** is a situation in which the monkey has the banana; that is, any state in which the last component is **has**:

  state(_, _, _, has).

22

PROLOG

Monkey and Banana (cont...)

- What are the allowed **actions/moves** that change the world from one state to another?
- **Allowed moves:**
  - **Grasp** banana
  - **Climb** box
  - **Push** box
  - **Walk** around
- **Not** all moves are possible in every possible state of the world.
  - Example: **grasp** is only possible if the monkey is standing on the box directly under the banana and does not have the banana yet.

23

PROLOG

Monkey and Banana (cont...)

- **Move** from one state to another
  move(State1, Action, State2)
  - **State1** is the state before the move.
  - **Action** is the action/move executed and
  - **State2** is the state after the move.
- The move '**grasp**', with its necessary precondition on the state before the move, can be defined by the clause:

```
move(state(middle,onbox,middle,hasnot),   % before move
     grasp,                               % action
     state(middle,onbox,middle,has)).     % after move
```

24

PROLOG

Monkey and Banana (cont...)

- In a similar way we can express the fact that the walk from any horizontal position P1 to any position P2.

  move(state(P1, onfloor, B, H),
  walk(P1,P2),
  state(P2, onfloor, B, H)).

- Note that this clause says many things, including, for example:
  - the action executed was 'walk from some position P1 to some position P2'
  - the monkey is on the floor before and after the move;
  - the box is at some point B which remained the same after the move;
  - the 'has banana' status (H) remains the same after the move.

25

PROLOG

Monkey and Banana (cont...)

- The other two types of actions, '**push**' and '**climb**', can be similarly specified.
- Example:
  - **Push:**

    move(state(P1, onfloor, P1, H),
    push(P1, P2),
    state(P2, onfloor, P2, H)).
  - **Climb:**

    move(state(P, onfloor, P, H),
    climb,
    state(P, onbox, P, H)).

26

PROLOG

Monkey and Banana (cont…)

- Main question our program will pose:

Can the monkey in some initial state get the banana?

- In terms of PROLOG predicate:

canget(State)

where the argument State is a state of the monkey world.

27

PROLOG

Monkey and Banana (cont…)

- The program for **canget(State)** can be based on two observations:

(1) For any state S in which the monkey already has the banana, the predicate **canget** must certainly be true; no move is needed in this case.

This corresponds to the PROLOG fact:

canget(state(_, _, _, has)).

28

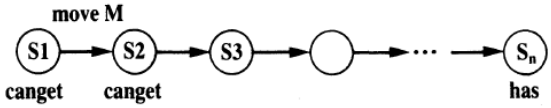
PROLOG

Monkey and Banana (cont…)

(2) In other cases one or more actions are necessary. The monkey can get the banana in any state (State1) if there is some move (Move) from State1 to some state (State2), such that the monkey can get the banana in State2 (in zero or more moves).

```
canget(State1) :-
    move(State1, Move, State2),
    canget(State2).
```

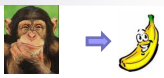
This principle can be illustrated as:



29

PROLOG

Monkey and Banana (cont…)

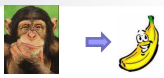


- Then the **Monkey and Banana program** is:

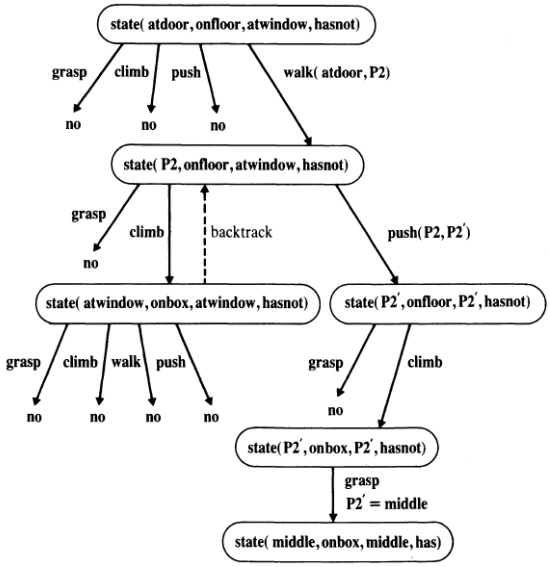
```
move(state(middle,onbox,middle,hasnot),
     grasp,
     state(middle,onbox,middle,has)).

move(state(P,onfloor,P,H),
     climb,
     state(P,onbox,P,H)).

move(state(P1,onfloor,P1,H),
     push(P1,P2),
     state(P2,onfloor,P2,H)).

move(state(P1,onfloor,B,H),
     walk(P1,P2),
     state(P2,onfloor,B,H)).

% change(State): monkey can get banana is State

canget(state(_,_,_,has)).          % can 1: Monkey already has it

canget(State1) :-                  % can2: Do some work to get it
    move(State1,Move,State2),      % Do something
    canget(State2).                % Get it now
```

PROLOG

Monkey and Banana (cont...)

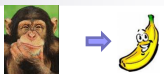


- **?- canget(state(atdoor,onfloor,atwindow,hasnot)).**



PROLOG

Order of Clauses and Goal



- Consider the following clause:

  p :- p.

- This says that "**p is true if p is true**".
- This is **declarative perfectly** correct but **procedurally is quite inoperable**.
- In fact, such a clause can cause problems to PROLOG.
- Consider the question:

  ? - p.

- Using the clause above, the goal **p** is replaced by the same goal **p**; this will be in turn replaced by **p**, etc.
- In such a case PROLOG will enter an **infinite loop!!!**

PROLOG



Order of Clauses and Goal (cont…)

- In our **monkey_and_banana** PROLOG program we have the following clause order:
  - Grasp
  - Climb
  - Push
  - Walk
- Effectively says that the monkey prefers grasping to climbing, climbing to pushing etc...
- This order of preferences helps the monkey to solve the problem.

But what could happen if the order was different?

33

PROLOG



Order of Clauses and Goal (cont…)

But what could happen if the order was different?

- Let assume that the new clause order is:

  Walk – Grasp – Climb – Push

- Then the execution of our original goal:

  ?- canget(state(atdoor, onfloor, atwindow, hasnot)).

  This results in an infinite loop!

- As the first move the monkey chooses will always be move, therefore he moves aimlessly around the room.
- **Conclusion:**
  - A program in PROLOG may be declaratively correct, but procedurally incorrect (i.e. Unable to find a solution when a solution actually exists).

34

PROLOG



Summary:

- The **matching operation** takes two terms and tries to make them identical by instantiating the variables in both terms.
- **Matching**, if it succeeds results in the most general instantiation of variables.
- The **declarative semantics** of PROLOG respect to a given program, and if variables it is true.
- A **comma** between goals means the conjunction of goals. A **semicolon** between goals means the disjunction of goals.
- The **procedural semantics** of PROLOG is a procedure for satisfying a list of goals in the context of a given program.
- The **declarative meaning** of programs in "pure" PROLOG doesn't depend on the order of clauses and the order of goals in clauses.
- The **procedural meaning** does depend on the order of goals and clauses. Thus the order can affect the efficiency of the program; an unsuitable order may even lead to infinite recursive calls.

35