SAPIENZA
UNIVERSITÀ DI ROMA

Artificial Intelligence: Prolog

Prof. Daniele Nardi and Prof. Luca Iocchi

2016/2017

Exercises: Logic programming and Prolog

Francesco Riccio

email: **riccio@dia.uniroma1.it**

Prolog

- **Prolog** is the major logic-based programming language (subset of First Order Logic)

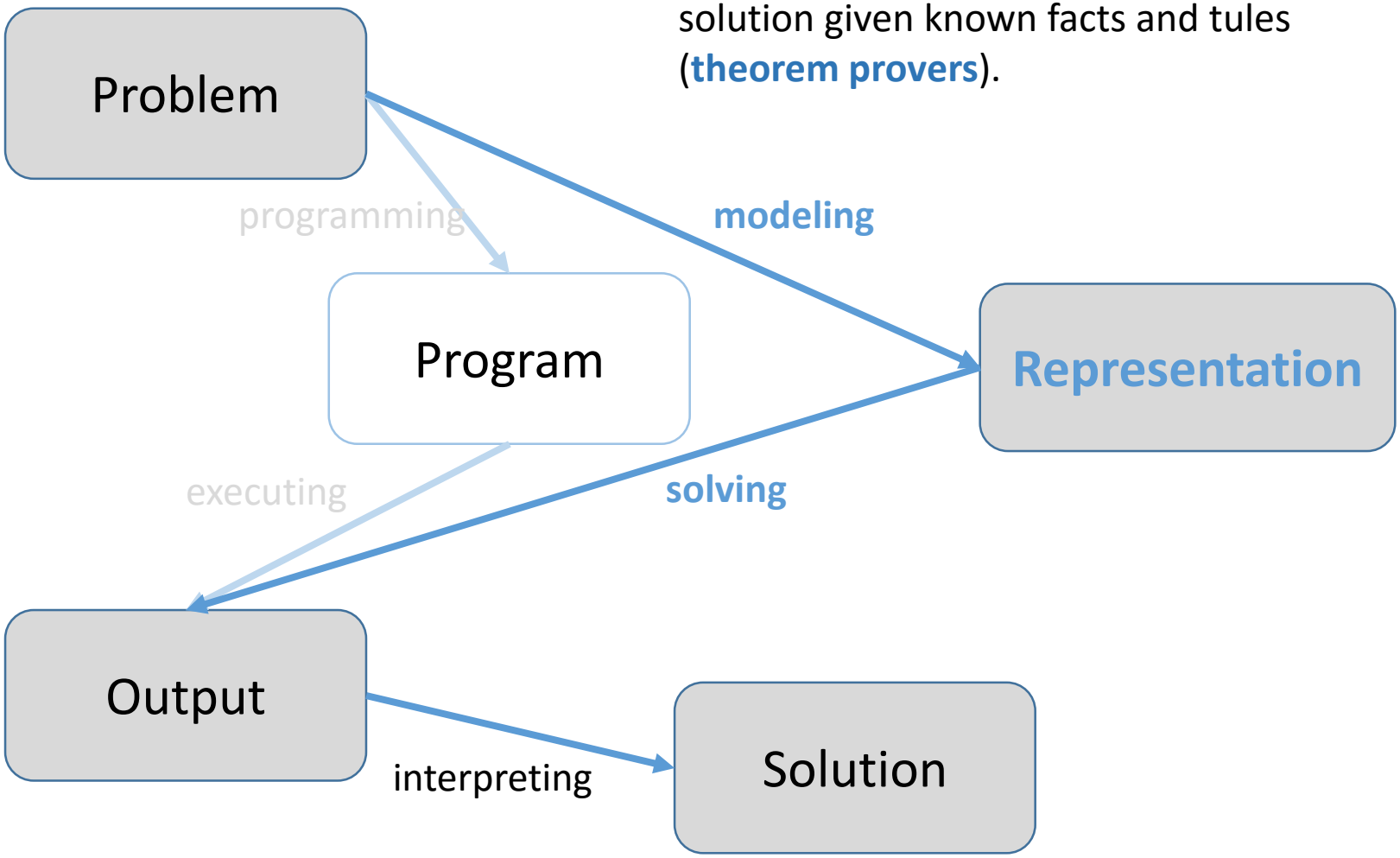
- Implementation and source code: **http://www.swi-prolog.org/**

- Textbooks:
  - L. Sterling, E. Shapiro, The Art of Prolog, 2nd Ed., Mit Press, 1994
  - http://www.learnprolognow.org/

Prolog

Logic Programming

1. **Definition** of the **problem** through the assertion of **facts** and **rules**.
2. **Querying** the system which **infers** the solution given known facts and tules (**theorem provers**).



Prolog

Aristotelic Syllogism ++

1) All men are mortal
2) Socrates is a man

⇒ We can infer that **Socrates is mortal**

KB:

```
mortal(X) :- man(X).
man(socrates).
```

⟷

```
mortal(X) :- man(socrates).
god(zeus).
```

Q: The inference is started by:

?- mortal(socrates) **?- mortal(zeus)**

Prolog

Logic Programming

```
grandfather(X, Z) :- father(X, Y), father(Y, Z).
grandfather(X, Z) :- father(X, Y), mother(Y, Z).
```

Son

```
son(X, Y) :- father(Y, X).
son(X, Y) :- mother(Y, X).
```

Parent

```
parent(P, X) :- father(P, X).
parent(P, X) :- mother(P, X).
```

Prolog

Logic Programming

```
father(daniele, michela).          mother(alma, eriberto).
father(daniele, jacopo).           mother(annamaria, daniele).
father(eriberto, daniele).         mother(annamaria, marcello).
father(antonio, eriberto).         mother(annamaria, sandro).

grandfather(X, Z) :- father(X, Y), father(Y, Z).
grandfather(X, Z) :- father(X, Y), mother(Y, Z).

nice(michela).          nice(anna).
```

KB

? - nice(michela)
? - grandfather(eriberto, X), nice(X)

? - nice(X)
? - grandfather(X, Z), nice(Z)

Q

Prolog

Abstract Interpreter

```
Input:   a goal G and a program P
Output:  an instance of G logical
         consequence of P if it exists,
         otherwise NO

begin
    R := G;          % R resolvent
    finished := false;

    Prove the goal in the resolvent;

    if R = { }
        then return G
        else return NO
end
```

```
while not R = { } and not finished do
begin
    choose a goal A in the resolvent;
    % renaming variables
    choose a clause A':- B1, …, Bn
    such that θ = unify(A, A');

    if no more choices
        then finished := true;
        else begin
                substitute
                A with B1, …, Bn in R;
                apply θ to R and G;
             end
end
```

Prolog

The Search Tree

- the root is the initial goal;
- every node has one successor for each clause whose head unifies with a goal in the node. Every successor has a resolvent obtained by the parent node by replacing the chosen goal with the body of the clause, after applying the unifier.

Every node contains a resolvent. If it is empty the node is a success node. A node without successors, not a success node, is a failure node.

Every success node represents a solution. If the tree cannot be further expanded and it does not have any success node then the goal fails.

Prolog

Recursive rule

```
father(daniele, michela).          mother(alma, eriberto).
father(daniele, jacopo).           mother(annamaria, daniele).
father(eriberto, daniele).         mother(annamaria, marcello).
father(antonio, eriberto).         mother(annamaria, sandro).

descendant(X, Y) :- son(X, Y).
descendant(X, Y) :- son(Z, Y), descendant(X, Z).

son(X, Y) :- father(Y, X).         son(X, Y) :- mother(Y, X).
```

KB

? - descendant(michela, eriberto).

Q

Prolog

The order matters

```
father(daniele, michela).          mother(alma, eriberto).
father(daniele, jacopo).           mother(annamaria, daniele).
father(eriberto, daniele).         mother(annamaria, marcello).
father(antonio, eriberto).         mother(annamaria, sandro).

descendant(X, Y) :- son(X, Y).
descendant(X, Y) :- son(Z, Y), descendant(X, Z).

descendant(X, Y) :- descendant(X, Z), son(Z, Y).   % 1 error example
descendant(X, Y) :- son(X, Y).   % 2 error example

son(X, Y) :- father(Y, X).         son(X, Y) :- mother(Y, X).
```

KB

```
? - descendant(daniele, X).
```

Q

Prolog

Ex: relatives

- Define the **relation son**; **parent**; **sibling**; **cousin**; **relation aunt/uncle**

```
father(daniele, michela).
father(daniele, jacopo).
father(eriberto, daniele).
father(antonio, eriberto).

son(X, Y) :- father(Y, X).
parent(P, X) :- father(P, X).

mother(alma, eriberto).
mother(annamaria, daniele).
mother(annamaria, marcello).
mother(annamaria, sandro).

son(X, Y) :- mother(Y, X).
parent(P, X) :- mother(P, X).

%---------------------------------
sibling(X, Y) :- son(X, P), son(Y, P), X \= Y.

cousin(X, Y) :- son(X, P1), son(Y, P2), sibling(P1, P2), X \= Y, P1 \= P2.

aunt_uncle(A, S) :-  son(S, P), sibling(P, A).
```

KB

Prolog

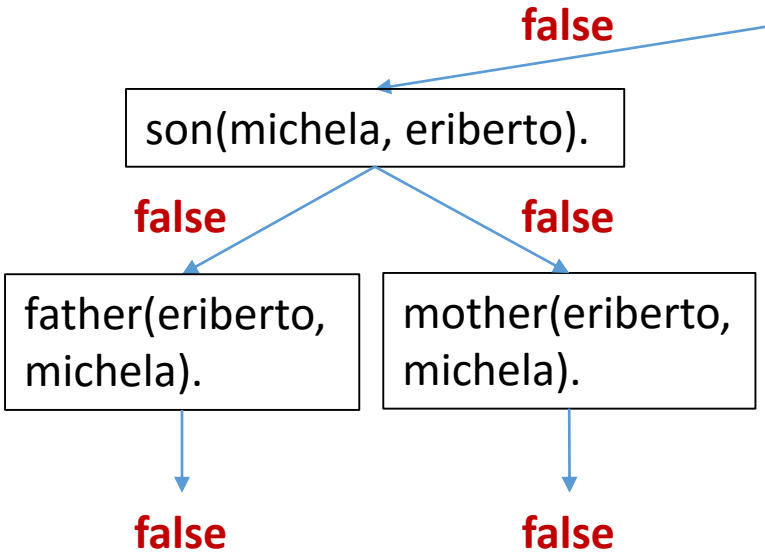
Ex: relatives

```
father(daniele, michela).          mother(alma, eriberto).
father(daniele, jacopo).           mother(annamaria, daniele).
father(eriberto, daniele).         mother(annamaria, marcello).
father(antonio, eriberto).         mother(annamaria, sandro).
son(X, Y) :- father(Y, X).         son(X, Y) :- mother(Y, X).

descendant(X, Y) :- son(X, Y).
descendant(X, Y) :- son(Z, Y), descendant(X, Z).
```

KB

- Build the **search tree** for the goal:

  ? - descendant(michela, eriberto).



Prolog

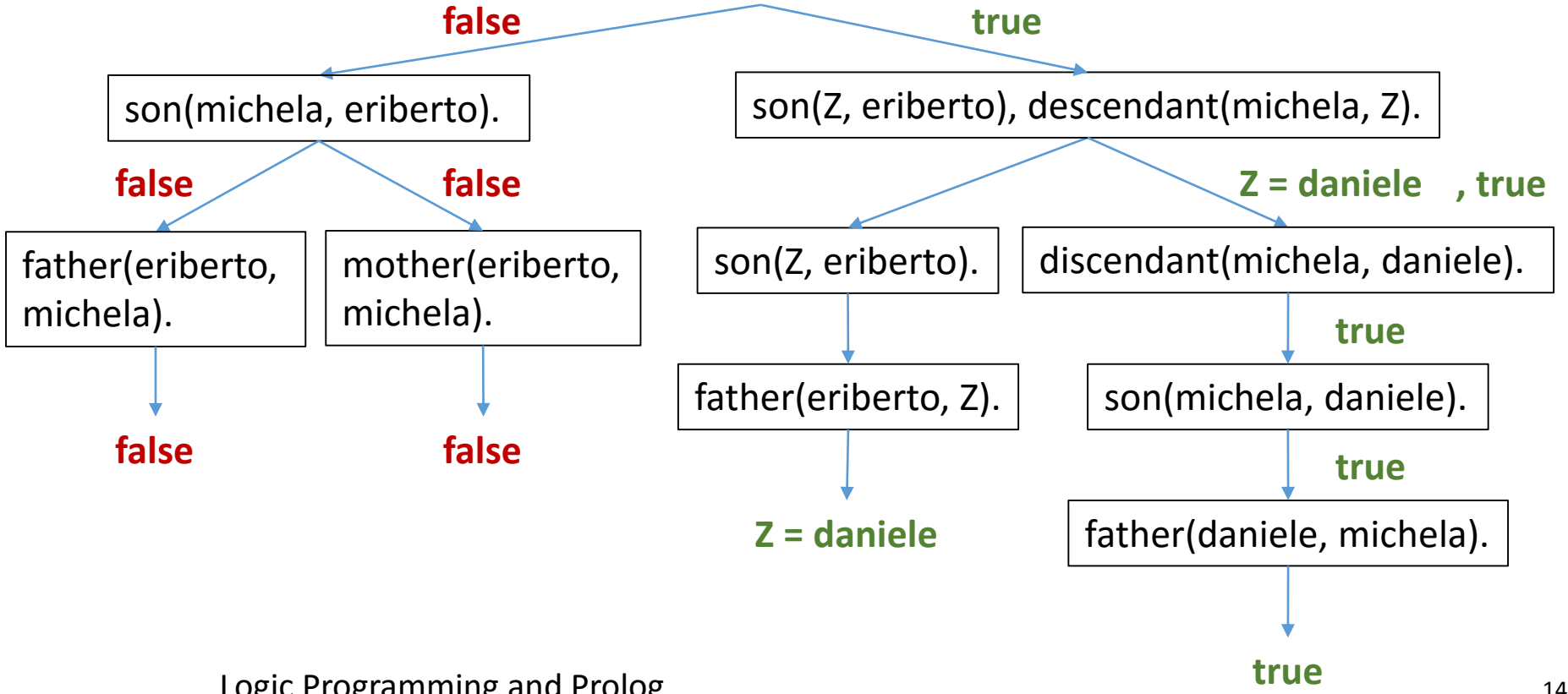
Ex: relatives

```
father(daniele, michela).
father(daniele, jacopo).
father(eriberto, daniele).
father(antonio, eriberto).
son(X, Y) :- father(Y, X).

mother(alma, eriberto).
mother(annamaria, daniele).
mother(annamaria, marcello).
mother(annamaria, sandro).
son(X, Y) :- mother(Y, X).

descendant(X, Y) :- son(X, Y).
descendant(X, Y) :- son(Z, Y), descendant(X, Z).
```

KB

- Build the **search tree** for the goal:

  ? - descendant(michela, eriberto).



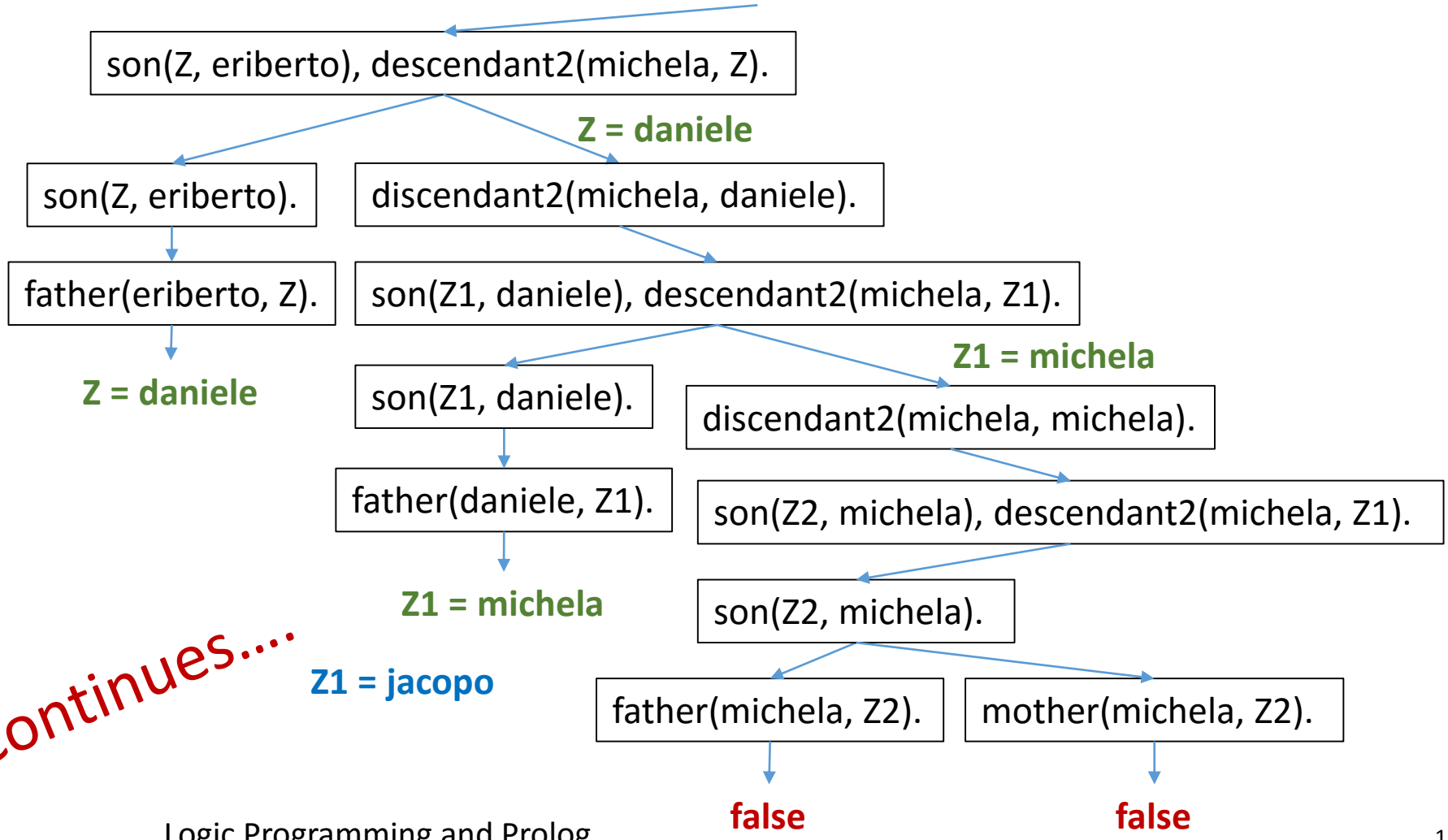
Prolog

Ex: relatives

descendant2(X, Y) :- son(Z, Y), descendant2(X, Z).
descendant2(X, Y) :- son(X, Y). **KB**

- check the **differences** with

? - descendant2(michela, eriberto).



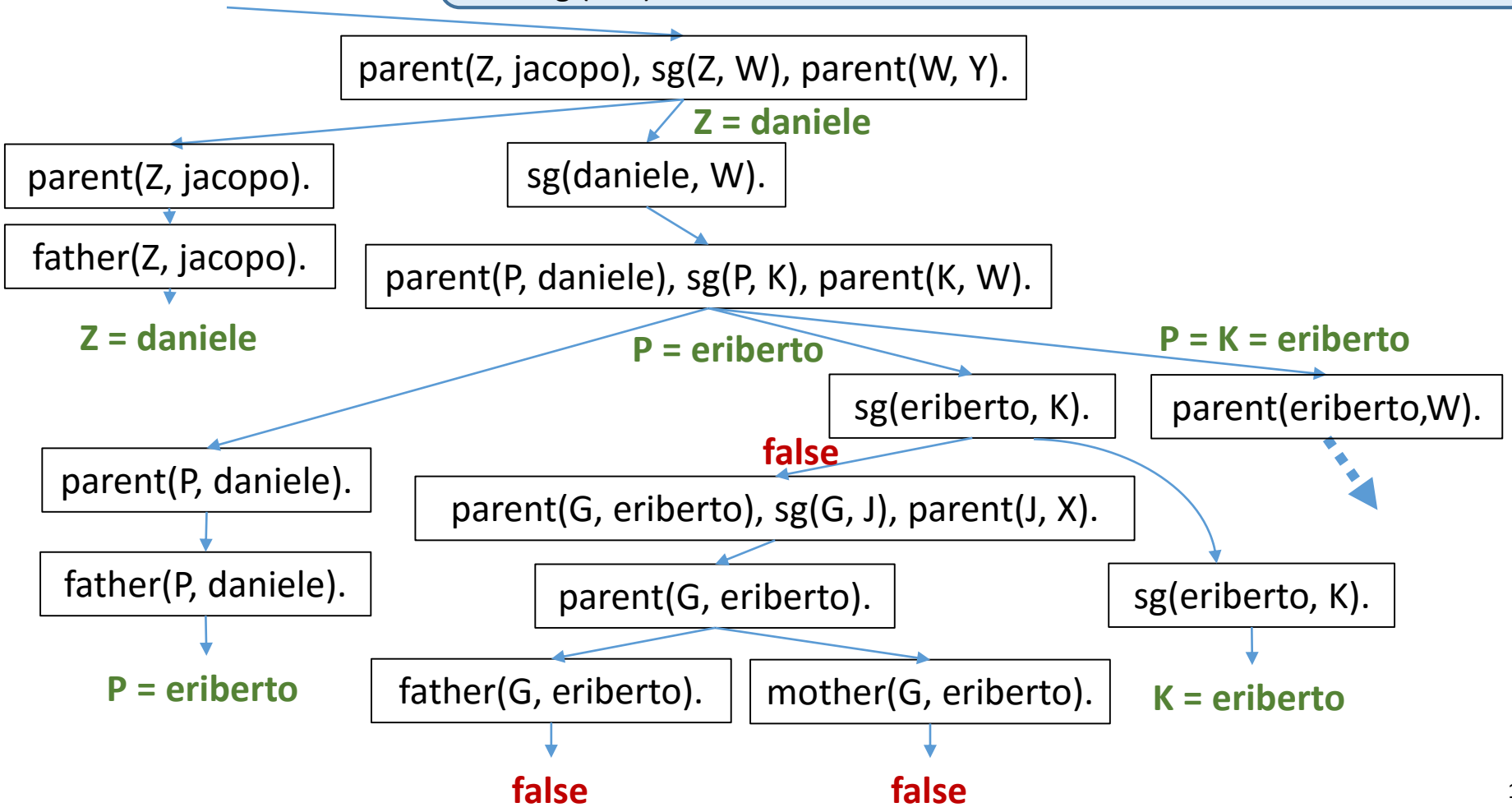
Prolog

Ex: relatives

- Build the **search tree** for the goal:
  ? - sg(jacopo, Y).

```
father(daniele, michela).
father(daniele, jacopo).
father(eriberto, daniele).
father(antonio, eriberto).
parent(P, X) :- father(P, X).

mother(alma, eriberto).
mother(annamaria, daniele).
mother(annamaria, marcello).
mother(annamaria, sandro).
parent(P, X) :- mother(P, X).

sg (X, Y) :- parent(Z, X), sg(Z, W), parent(W, Y).
sg (X, X).
```

KB



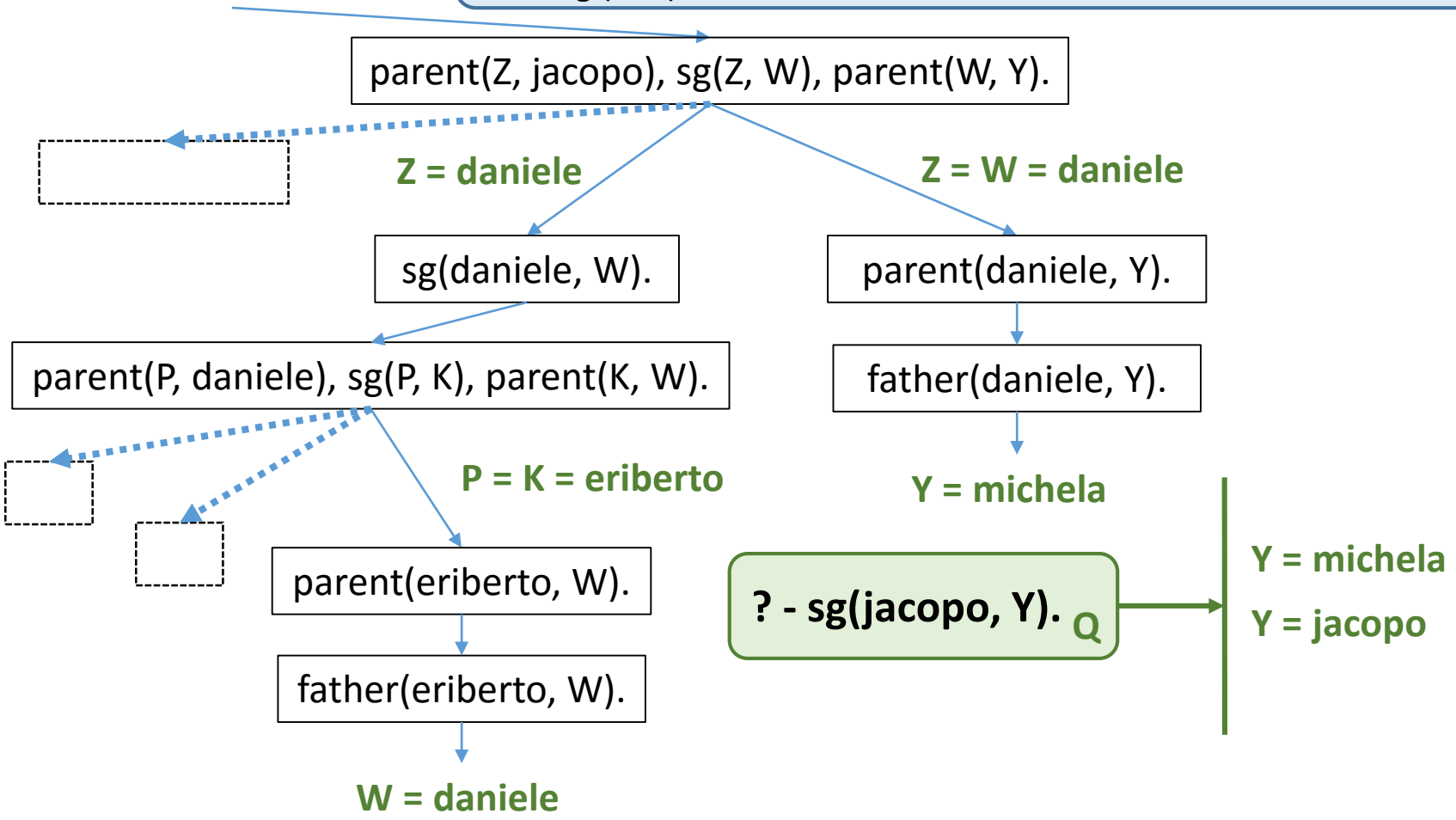
Prolog

Ex: relatives

- Build the **search tree** for the goal:
  ? - sg(jacopo, Y).

```
father(daniele, michela).          mother(alma, eriberto).
father(daniele, jacopo).           mother(annamaria, daniele).
father(eriberto, daniele).         mother(annamaria, marcello).
father(antonio, eriberto).         mother(annamaria, sandro).
parent(P, X) :- father(P, X).      parent(P, X) :- mother(P, X).

sg (X, Y) :- parent(Z, X), sg(Z, W), parent(W, Y).
sg (X, X).
```

KB



Prolog

Ex: Matryoshka dolls

- Write a knowledge base representing which doll **is directly** contained in which other doll.
- Write a recursive predicate **in/2**, that tells us which doll **is contained** in which other dolls.

```
in(X, Y) :- in(X, Z), in(Z, Y).
```

KB

? - in(olga, irina). **? - in(natasha, katarina).** **? - in(X, katarina).**

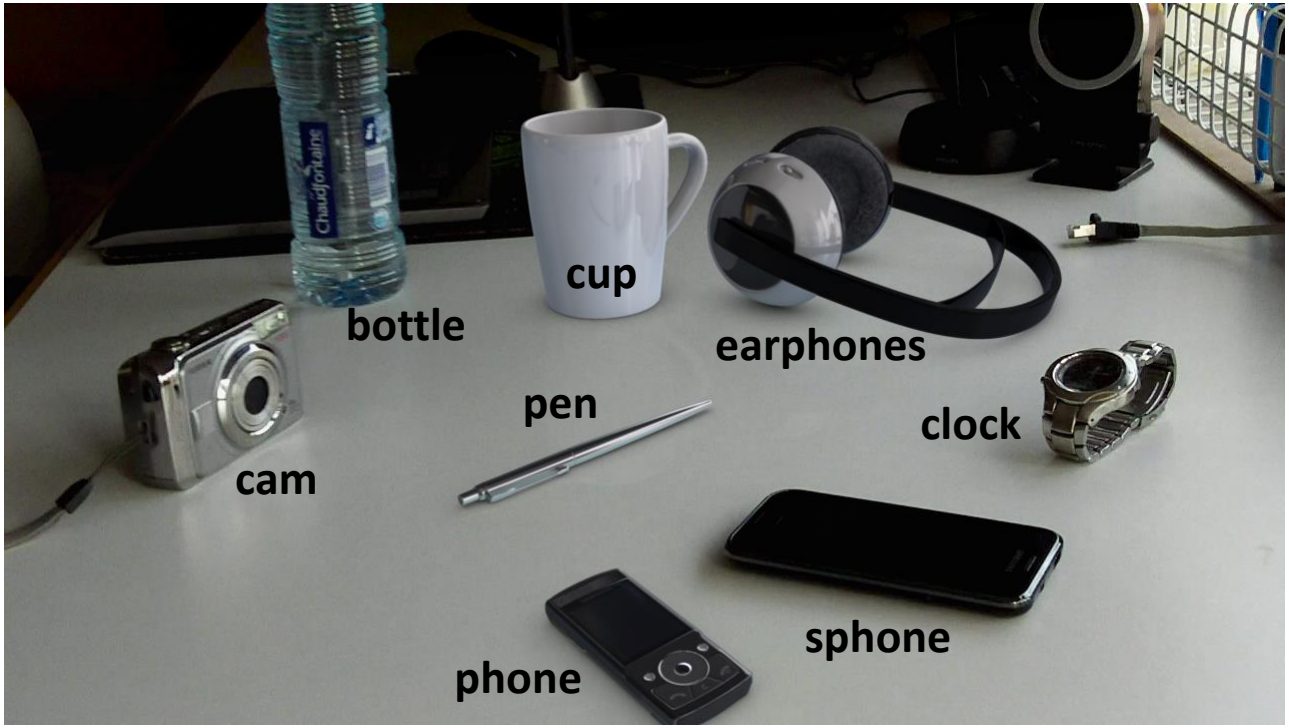
Q

Prolog

Ex: Relative Spatial Relations

Define the **relative spatial relations** needed to represent the given scenario:

left, right, front, behind, between, on, below




```
left(bottle, cup).
left(cup, earphones).
left(cam, pen).
left(pen, sphone).
left(phone, sphone).
left(sphone, clock).
left(earphones, clock).
...
```

KB

? - left(Obj, clock). **? - front(Obj, phone).** **? - behind(pen, Obj).**

Q

Prolog

IS predicate

Is is a predicate, true when the *evaluation* of the expression B returns a value, that is **assigned** to the variable A.

Predicates defined using is are **not invertible**:

? - 3 is 5 + Y

Does **not assignes** a value to Y such that the predicate is true.

Prolog

Ex: Compute the factorial number

The **factorial** of a non-negative integer n, denoted by **n!**, is the product of all positive integers less than or equal to n.

Factorials

$n! = n(n-1)(n-2)\ldots 1$

$0! \equiv 1$ (by definition)

$1! = 1$

$2! = 2 \times 1 = 2$

$3! = 3 \times 2 \times 1 = 6$

```
factorial(0, 1).
factorial(Y, X) :- Y > 0, Y1 is Y-1,
                          factorial(Y1, X1),
                          X is Y*X1.
```

KB

? - factorial(1, 1). ? - factorial(4, F). ? - factorial(Num, 1).

Q

SWI Prolog **+ GUI**

```
1 ?- guitracer.
% The graphical front-end will be used for subsequent tracing
true.

2 ?- trace.
true.

[trace] 2 ?-
```

Prolog

Ex: Multiplication and Power

```
multiplication(_, 0, 0).
multiplication(0, _, 0).
multiplication(X, Y, M) :- Y > 0,
        Y1 is Y - 1,
        multiplication(X, Y1, M1),
        M is M1+X.
```

KB

? - multiplication(3, 4, M).

Q

multiplication/3 ?

```
power(_, 0, 1).
power(X, Y, P) :- Y > 0, Y1 is Y - 1,
        power(X, Y1, P1), P is P1*X.
```

KB

? - power(2, 4, P).

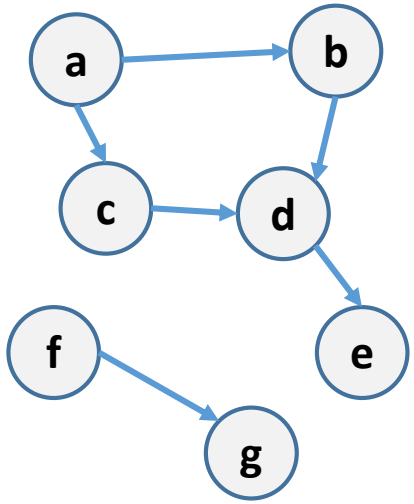
Q

Prolog

Directed Graph

```
arc(a, b).      arc(a, c).
arc(b, d).      arc(c, d).
arc(d, e).      arc(f, g).

connected(X, X).
connected(X, Y) :- arc(X, Y).
connected(X, Y) :- arc(X, Z), connected(Z, Y).
```

KB

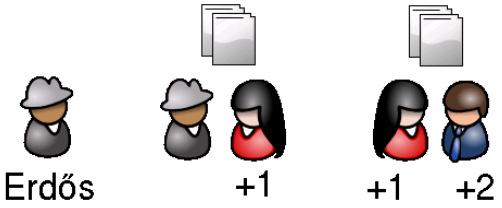


? - connected(a, c) **? - connected(a, d)**
? - connected(g, a) **? - connected(X, Y)**

Q

Prolog

Ex: Erdős number



The **Erdős number** is the **number** of *arcs* needed to connect the author of a paper with the mathematician Paul Erdős.
An author's Erdős number is: **1** if he has co-authored a paper with Erdős; **2** if he has co-authored a paper with someone who has co-authored a paper with Erdős; etc…

```
publication(nardi, brachman).
publication(brachman, erdos).
publication(einstein, erdos).
publication(konolige, erdos).
publication(september, nardi).

erdos_num(X, 1) :- publication(X, erdos).
erdos_num(X, Xn) :- publication(X, Y), erdos_num(Y, Yn), Xn is Yn + 1.
```

KB

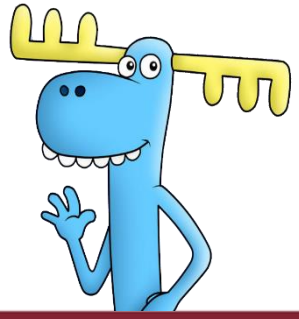
? - erdos_num(nardi, En). **? - erdos_num(Author, 1).** **? - erdos_num(Author, En).** Q

Prolog

Ex: Friendship

- Write knowledge base that
  represents your **friendship relations**;

- Write a prolog program that **counts** the number of your:
  - friends;
  - female friends;
  - male friends;
  - friends living in your city;



Prolog

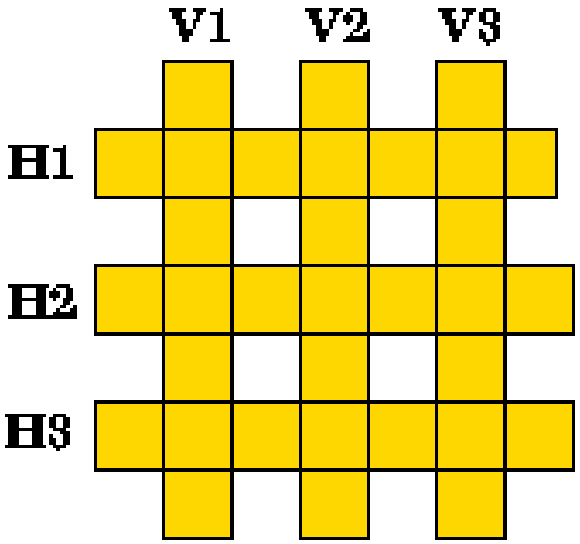
More exerises on http://www.learnprolognow.org/

Ex: Crossword

```
word(astante,  a,s,t,a,n,t,e).
word(astoria,  a,s,t,o,r,i,a).
word(baratto,  b,a,r,a,t,t,o).
word(cobalto,  c,o,b,a,l,t,o).
word(pistola,  p,i,s,t,o,l,a).
word(statale,  s,t,a,t,a,l,e).

crossword(V1, V2, V3, H1, H2, H3) :-
        word(V1, _, V12, _, V14, _, V16, _),
        word(V2, _, V22, _, V24, _, V26, _),
        word(V3, _, V32, _, V34, _, V36, _),

        word(H1, _, H12, _, H14, _, H16, _),
        word(H2, _, H22, _, H24, _, H26, _),
        word(H3, _, H32, _, H34, _, H36, _),

        V12 = H12, V22 = H14, V32 = H16,
        V14 = H22, V24 = H24, V34 = H26,
        V16 = H32, V26 = H34, V36 = H36.
```

KB

- Write prolog program that solves a simplified version of the **crossword problem**.
- The yellow intersection-squares must contain a character which is **exactly the same** for both the vertical and horizontal word.



SAPIENZA
UNIVERSITÀ DI ROMA

Artificial Intelligence: Prolog

Prof. Daniele Nardi and Prof. Luca Iocchi

2016/2017

Exercises: Logic programming and Prolog

Francesco Riccio

email: **riccio@dia.uniroma1.it**