DATA MODELLING IN ENGINEERING

MODELING BASED ON LOGIC PROGRAMMING – PART II -

Ana Inês Oliveira aio@fct.unl.pt

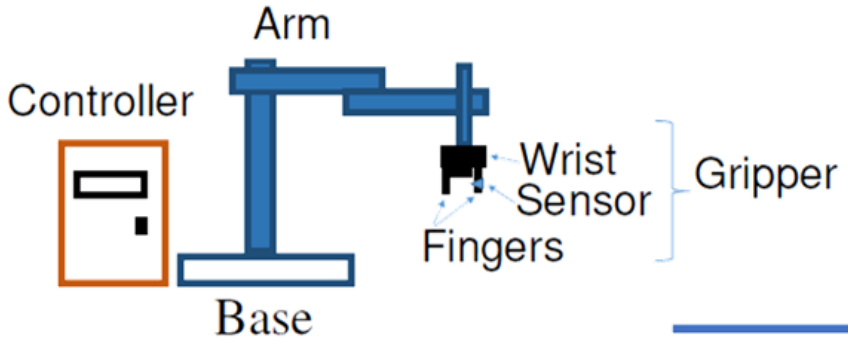
2024 - 2025

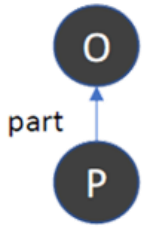
Contents

- **PROLOG**
  - Representation of Facts
  - Unification
  - Representation of Rules
  - Representation of Queries
  - Backtracking mechanism
  - Recursion mechanism
  - Facts / Rules / Queries
  - Structures
  - Combined Queries
  - Arithmetic
  - Changing the memory of PROLG
  - INPUT / OUTPUT

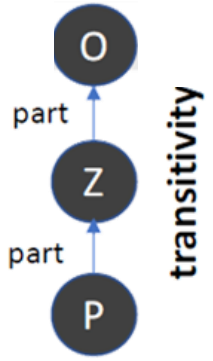
Facts / Rules / Queries

Going back to the robot model example:
We can generalize the rule





includes(O,P) :- part(O,P). /* O includes P **if** O has a part P */
includes(O, P) :- part(O,Z), part(Z,P). /* O includes P **if** O has a part Z **and** Z has a part P*/



<u>One possible solution</u>:

```
part(robot, base).
part(robot, arm).
part(robot, gripper).
part(robot, controller).
part(gripper, wrist).
part(gripper, fingers).
part(gripper, sensor).
```

A more generic solution

contains(O,P) :- part(O,P). /* O contains P **if** O has part P */
contains(O,P) :- part(Z,P), **contains(O,Z)**. /* O contains P **if** Z has part P **and** O contains Z */

The 2nd rule is defined in t
... i.e. recursive definition

recursion mechanism

Structures

PROLOG enables the representation of complex information in a way that's natural for logic programming

facts

```
/*       #id,      name, birthdate    */
person(31267389, john,   birthdate(24,11,2000)).
person(43261876, mary,   birthdate(16,06,2001)).
person(37392715, thomas, birthdate(05,03,2000)).
...
```

Example of structure

Example of Queries:

?- person(_, N, birthdate(16,06,2001)).

N = mary.

?- person(Nr, _, birthdate(05,03,2000)).

Nr = 37392715.

?- person(37392715, _, birthdate(Day,Month,Year)).

Day = 5,
Month = 3,
Year = 2000.

Structures

```
/*      #id,      name, age, birthdate, address  */
person(31267389, john,   birthdate(24,11,2000), address('R Bernardo Marques', 7, 'Caprica')).
person(43261876, mary,   birthdate(16,06,2001), address('R Francisco Costa', 5, 'Caprica')).
person(36482754, jane,   birthdate(30,01,2004), address('R Garcia de Orta', 3, 'Almada')).
person(37392715, thomas, birthdate(05,03,2000), address('R Alfredo Cunha', 9, 'Caprica')).

...
```

Identify by name, people that live in Almada:

?- person(_, N, _, address(_, _, 'Almada')).

N = jane.

Exercises: Write Prolog queries to:

1. Identify the **name** and **birthdate** of people who live in Caparica.
2. Identify **two** people (by **ID number**) who live in the same city.

Combined Queries

PROLOG enables the combined queries that refer to queries that involve multiple goals, joined using logical connectives like conjunctions (,) and disjunctions (;).

```
/*          Nr, Name, gender, year     */
student(65200, 'Ademir Paulo Santos Caetano',m,3).
student(65145, 'Afonso Aleixo Vieira Gonçalves Varela',m,3).
student(65405, 'André Filipe Freitas Mendes',m,3).
student(65368, 'André Gomes Antunes',m,3).
student(54543, 'António Manuel Sebastião Ferreira Deveza',m,3).
student(65499, 'Beatriz Flórido Pereira',f,3).
student(65243, 'Bernardo de Oliveira Lopes Garcia Barata',m,3).
% ...
student(66092, 'Vasco Cananão Mendes',m,3).

gender(f, female).
gender(m, male).
```

What is the gender of student nº 65405 ?

?- student(65405,_,G,_) , gender(G,Gender).

and

G = m , Gender = male.

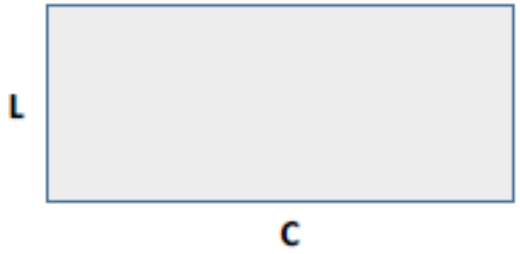
What are the numbers female students ?

?- gender(G,female) , student(N,_,G,_).

G = f , N = 65499.

Arithmetic

+ - * / ... is




```
area(L,C,A) :- A is L * C.
perimeter(L,C,P) :- P is 2*(L+C).
```

```
?- area(20, 4, A).
A = 80
?-area(20, 4, 80).
true
```

Understanding **"is"**

```
?- A is 3 + 1.
A = 4
?- 4 is 3 + 1.
true
```

Other operations: ** or ^ (power), // (integer div), abs(...), sin(...), cos(...), tan(...), ...

```
interval(X,A,B) :- X >= A, X =< B.
```

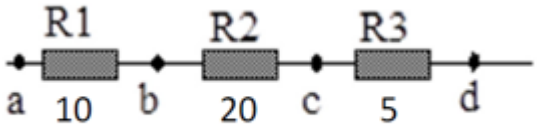
See: http://www.swi-prolog.org/pldoc/man?section=funcsummary

https://www.swi-prolog.org/pldoc/man?section=funcsummary

Rules & Recursivity …. more

Example:

Resistive Circuit (serial)



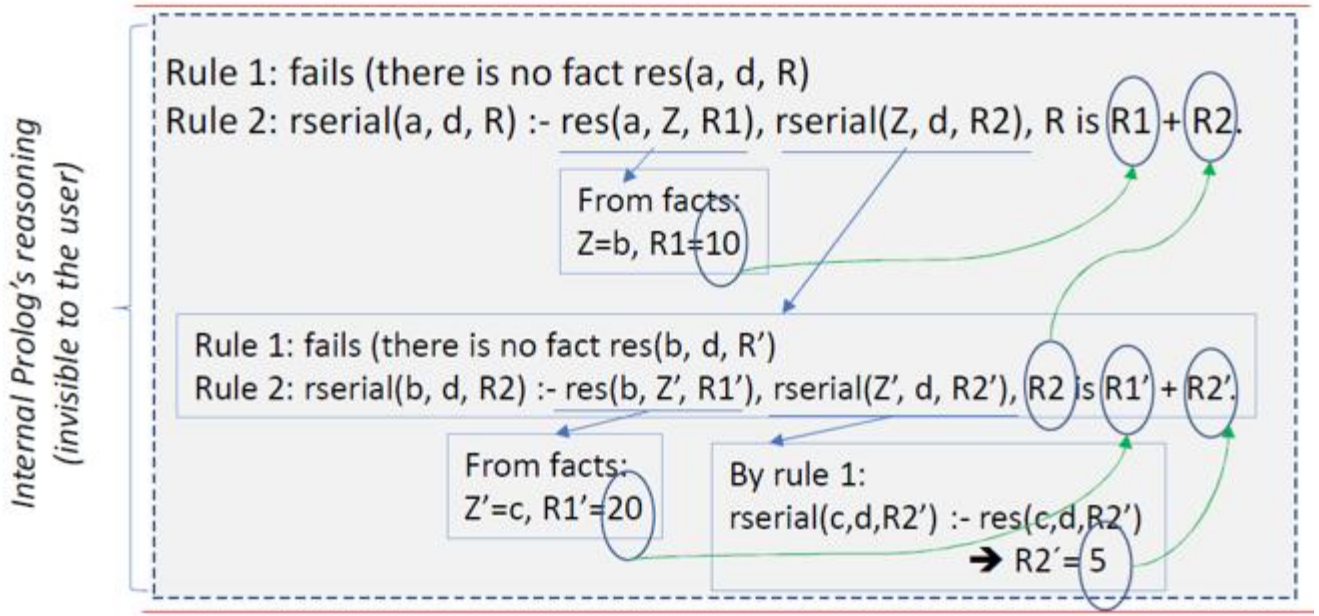

```
res(a,b,10).
res(b,c,20).
res(c,d,5).
```

recursive definition

R1 `rserial(X,Y,R) :- res(X,Y,R).`

R2 `rserial(X,Y,R) :- res(X,Z,R1), rserial(Z,Y,R2), R is R1 + R2.`

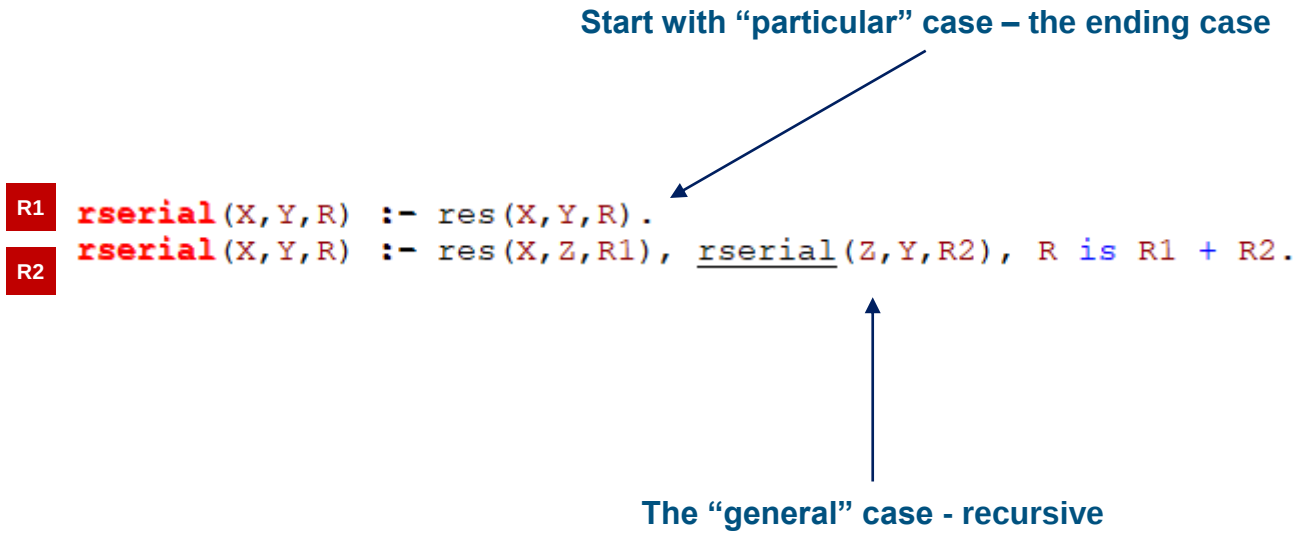
```
[1]  ?- rserial(a,d,R).
R = 35 .
```



from @L.M. Camarinha-Matos 2023

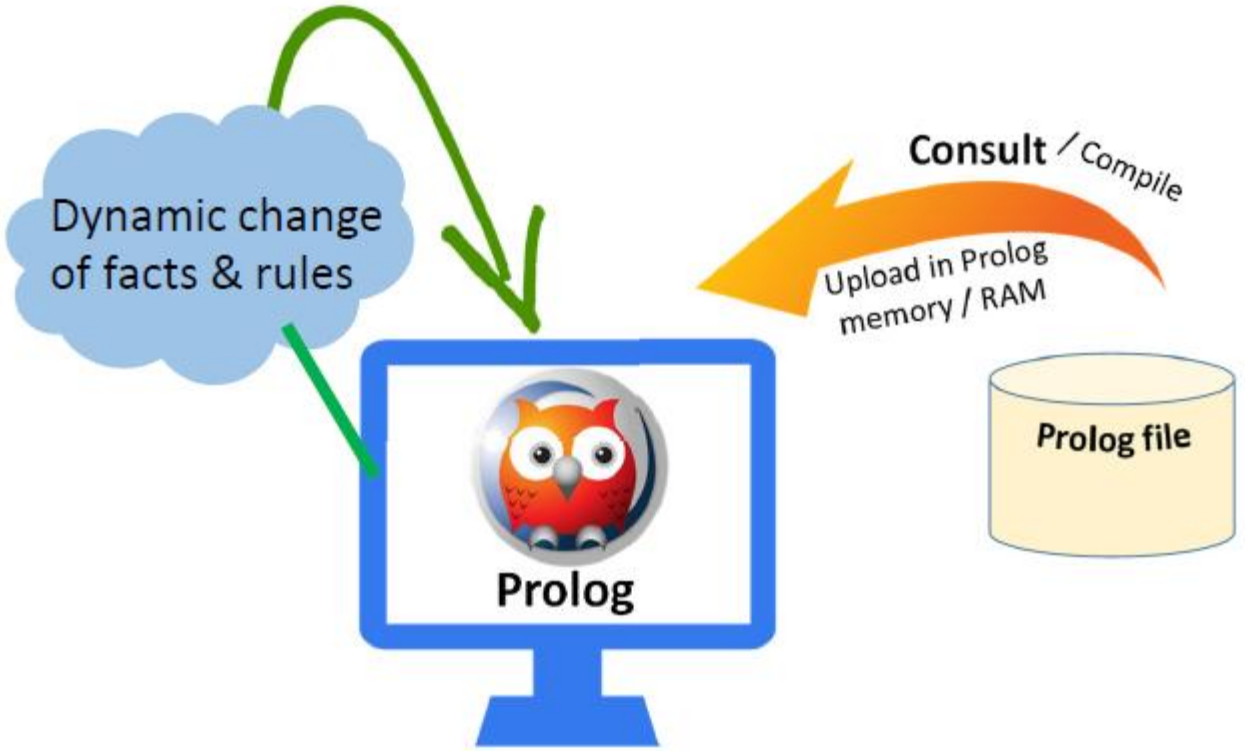
Rules & Recursivity …. more

Note on recursive rules:



from @L.M. Camarinha-Matos 2023

Changing the Memory in Prolog



Changing the Memory in Prolog

Dynamic change of memory

```
asserta(+Term)
Assert a fact or clause in the database. Term is asserted as the first fact or clause of the
corresponding predicate. Equivalent to assert/1, but Term is asserted as first clause or fact of the
predicate.

assertz(+Term, -Reference)
Equivalent to asserta/1, asserting the new clause as the last clause of the predicate.

retract(+Term)
When Term is an atom or a term it is unified with the first unifying fact or clause in the database.
The fact or clause is removed from the database.

retractall(+Head)
All facts or clauses in the database for which the head unifies with Head are removed. If Head refers
to a predicate that is not defined, it is implicitly created as a dynamic predicate. See also
dynamic/1.35
```

https://en.wikibooks.org/wiki/Prolog/Modifying_the_Database

assert(S)
asserta(S)
assertz(S)
} *To add facts / rules*

retract(S)
} *To remove/delete facts / rules*

Imagine we have defined the facts:

...
father(manuel, luis).
father(luis, jose).
...

We can add a new fact to the beginning of the set:

?-**asserta**(father(antonio, manuel))

Or to the end of the set:

?-**assertz**(father(jose, maria))

If we use:

?-**assert**(father(luis, ana)) → Adds it in any random position in the set (in some implementations at the end)

from @L.M. Camarinha-Matos 2023

Changing the Memory in Prolog

ASSERT

```
?- assert(father(manuel,luis)).
true.

?- assertz(father(luis,jose)).
true.

?- listing(father).
father(manuel, luis).
father(luis, jose).
true.

?- asserta(father(antonio,manuel)).
true.

?- listing(father).
father(antonio, manuel).
father(manuel, luis).
father(luis, jose).
true.
```

```
?- assertz(father(jose,maria)).
true.

?- listing(father).
father(antonio, manuel).
father(manuel, luis).
father(luis, jose).
father(jose, maria).
true.

?- asserta(father(carlos,antonio)).
true.

?- assert(father(carlos, clara)).
true.
```

```
?- listing(father).
father(carlos, antonio).
father(antonio, manuel).
father(manuel, luis).
father(luis, jose).
father(jose, maria).
father(luis, ana).
father(carlos, clara).
true.
```

Changing the Memory in Prolog

DYNAMIC

If we use **assert** in a program to add (in run-time) facts (or rules) for which we don't have any with the same structure, some compilers may "complain" ....

That is the case o SWI-Prolog.
To avoid this problem, we can give an instruction to the compiler:

:- **dynamic** father/2.

In this way, the compiler will take into account that facts of the form father(_,_), i.e. with 2 parameters, might be added dynamically in run time

Example: Program to acquire a sequence of facts in the form 'father(X,Y)' ended by the word 'end'.

```
:- dynamic father/2.

read_fathers :- read(S), memorize(S).

memorize(end).
memorize(father(X,Y)) :- assertz(father(X,Y)), nl, read_fathers.
memorize(_) :- write( ' => Invalid data'), nl, read_fathers.
```

Here we use some pre-defined rules of SWI-Prolog:
read – reads a string ended by "."
write – writes a string
nl – new line

from @L.M. Camarinha-Matos 2023

Changing the Memory in Prolog

RETRACT

Imagine we have the following facts in memory:

```
father(carlos, antonio).
father(antonio, manuel).
father(manuel, luis).
father(luis, jose).
father(jose, maria).
father(luis, ana).
father(carlos, clara).
```

```
?- retract(father(jose,maria)).
true.

?- listing(father).
:- dynamic father/2.

father(carlos, antonio).
father(antonio, manuel).
father(manuel, luis).
father(luis, jose).
father(luis, ana).
father(carlos, clara).

true.
```

```
?- retract(father(luis,X)).
X = jose ;
X = ana.

?- listing(father).
:- dynamic father/2.

father(carlos, antonio).
father(antonio, manuel).
father(manuel, luis).
father(carlos, clara).

true.
```

Changing the Memory in Prolog

RETRACT

Example: Program to delete all facts of the form 'father(X,Y)'.

```
delete_fathers :- retract(father(_,_)), fail.
```

*Please note the use of the pre-defined predicate **fail** in order to guarantee that, through the mechanism of "backtrack", all facts 'father' are deleted.*

*The **fail** predicate always fails*

If we ask:

```
?- delete_fathers.
false.
```

But the memory is empty:

```
?- listing(father).
:- dynamic father/2.

true.
```

How to avoid this answer?

Changing the Memory in Prolog

RETRACT

We could add a 2nd rule that is only executed after the first rule removes all "fathers" and guarantees that the rule succeeds:

```
delete_fathers :- retract(father(_,_)), fail.
delete_fathers.
```

```
?- read_fathers.
|: father(antonio,carlos).

|: father(carlos,luis).

|: father(luis,ana).

|: end.

true .
```

```
?- listing(father).
:- dynamic father/2.

father(antonio, carlos).
father(carlos, luis).
father(luis, ana).

true.
```

```
?- delete_fathers.
true.

?- listing(father).
:- dynamic father/2.

true.
```

The 2nd rule is only executed when retract fails (i.e. there are no more "fathers" to delete

The 2nd rule simply succeeds

Changing the Memory in Prolog

RETRACT

```
delete_fathers:-retract(father(_,_)),fail.
delete_fathers.
```

…..or use

retractall(father(_,_)).

```
delete_fathers:-retractall(father(_,_)).
```

Input / Output

How can we rewrite this code to avoid ending input with .

Exercise

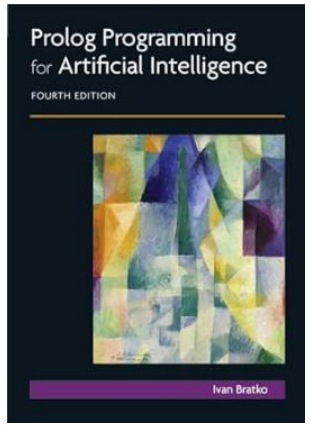
Let's get back to the example of fathers and create a menu for a “FATHERS MANAGEMENT SYSTEM ☺”

```
gmenu:- nl,nl,write('FATHERS MANAGEMENT SYSTEM :)'),nl,
    menu(Op), execute(Op).

menu(Op):- write('1. List fathers'),nl,
    write('2. Insert father'),nl,
    write('3. Delete fathers'),nl,
    write('4. Exit'), nl, readoption(Op).

readoption(Op):- read(Op),valid(Op),nl.
readoption(Op):- nl, write('*** Invalid option. Try again: '), readoption(Op).

valid(Op):- Op >=1, Op=<4.

execute(4). % exit condition
execute(Op):- exec(Op),nl,
    menu(NOp),execute(NOp).

exec(1) :- listing(father).
exec(2) :- read_fathers.
exec(3) :- delete_fathers.
```

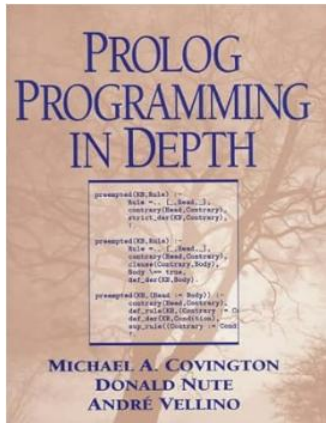
FATHERS MANAGEMENT SYSTEM :)
1. List fathers
2. Insert father
3. Delete fathers
4. Exit

Here we use some pre-defined rules of SWI-Prolog:
read – reads a string ended by “.”
write – writes a string
nl – new line

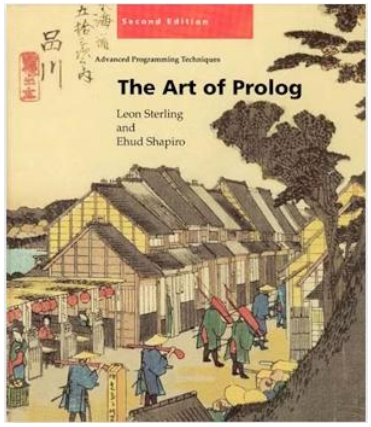
Further reading



https://www.amazon.com/Programming-Artificial-Intelligence-International-Computer/dp/0321417461



https://www.amazon.com/Prolog-Programming-Depth-Michael-Covington/dp/013138645X/ref=pd_sim_14_4?ie=UTF8&dpID=514M0RXA1WL&dpSrc=sims&preST=_AC_UL160_SR122%2C160_&refRID=1TM7A3CEFC2BD4JA77WR



https://mitpress.mit.edu/9780262691635/the-art-of-prolog/

(...)

https://www.swi-prolog.org/pldoc/doc_for?object=manual

https://en.wikibooks.org/wiki/Prolog

https://drsmithbiology.weebly.com/further-reading.html

Good Work!

Ana Inês Oliveira

NOVA School of Sciences and Technology | FCT NOVA

aio@fct.unl.pt