NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

MDE

Modeling with PROLOG
- Part II -

Ana Inês Oliveira

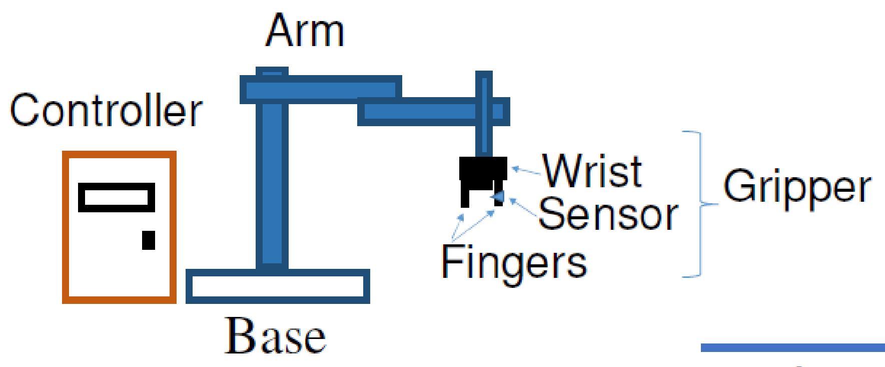
Nova University of Lisbon
School of Science and Technology

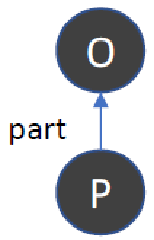
aio@fct.unl.pt or aio@uninova.pt

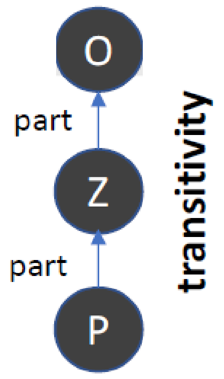
CONTENTS

FACTS / RULES / QUERIES

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





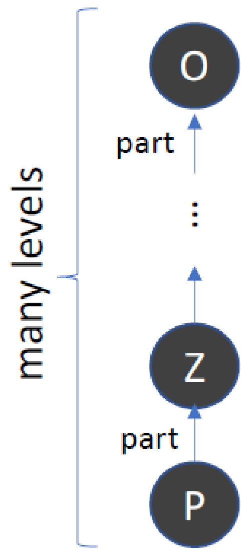


includes(O,P) :- part(O,P). /* O includes P if O has a part P */
includes(O, P) :- part(O,Z), part(Z,P). /* O includes P if O has a part Z and Z has a part P*/

One possible solution:

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

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

recursion mechanism

STRUCTURES

Example of **structure**

facts

```
order(305, date(11,10,2022), p45, 20).    /* # order, date, product, quantity */
order(125, date(1,5,2022), p34, 5).
order(235, date(4,2,2023), p34, 16).
...
```

The name of the structure ('date' in this case) is chosen by us, not a pre-defined keyword in Prolog

Example queries:

```
?-order(_, D, p34,_).
 D = date(1, 5, 2022)
?-order(E, date(1,5,2022), _, _).
E = 125
?-order(235, date(Day, Month, Year), _, _).
Day = 4,
Month = 2,
Year = 2023
?-order(_, date(1, 5, 2022), P,_).
P = p34
```

STRUCTURES

```
/* # order, date, product, quantity, delivery address*/

order(305, date(11,10,2022), p45, 20, delivery('R Raul Brandão, 5', 'Almada')).
order(125, date(1,5,2022), p34, 5, delivery('R Fernando Simões, 12', 'Caparica')).
order(235, date(4,2,2023), p34, 16, delivery('R Raul Brandão, 17', 'Almada')).
...
```

Identify orders to be delivered in Almada:

```
?- order(N, _, _, _, delivery(_, 'Almada’)).
N = 305  ;
N = 235
```

Exercises: Write the Prolog queries for:

1. Which product and which quantity is to be delivered in Caparica?
2. Identify 2 products (product id) to be delivered in the same city

COMBINED QUERIES

```
student(52417, 'Afonso Maria', m, 2).
student(52828, 'Alessia Offsas', f, 3).
student(53202, 'Alexandre Cardoso', m, 2).
student(52431, 'Alexandre Brito’, m, 3).
student(52993, 'Alexandru Botnari', m, 3).
student(52418, 'Americo Alves', m, 3).
student(51789, 'Ana Rita Silva', f, 2).
...
student(52751, 'Waner Shan', f, 3).


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

What is the gender of student nº 52993?

```
?-student(52993, _, G, _), gender(G, Gender).
Gender = male
```

and

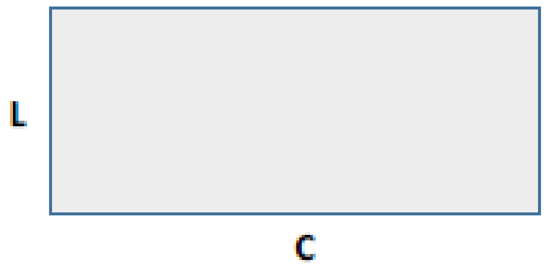
Who is a female student?

```
?-gender(G, female) , student(N, _, G, _).
N = 52828 ;
N = 51789 ;
...
```

Meaning: try to find a different answer

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
```

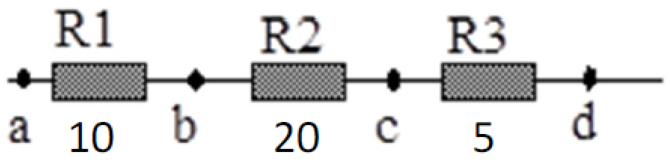
Other operations: ** or ^ (power), // (integer div), abs(...), sin(...), cos(...), tan(...), ...

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

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

RULES & RECURSIVITY …. more

Resistive circuit (serial)



***Recursive** definition*

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

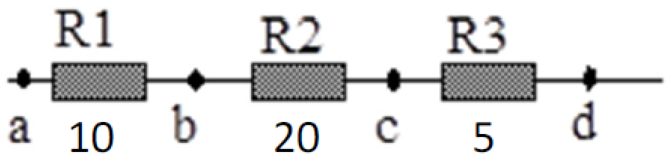
R2 **rserial(X,Y,R) :- res(X,Z,R1), rserial(Z,Y,R2), R is R1 + R2.**

```
?-rserial(a, d, R).
```

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

RULES & RECURSIVITY …. more

Resistive circuit (serial)



One solution:

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.**

?-rserial(a, d, R).

Internal Prolog's reasoning (invisible to the user)

Rule 1: fails (there is no fact res(a, d, R)
Rule 2: rserial(a, d, R) :- res(a, Z, R1), rserial(Z, d, R2), R is R1 + R2.

From facts:
Z=b, R1=10

Rule 1: fails (there is no fact res(b, d, R')
Rule 2: rserial(b, d, R2) :- res(b, Z', R1'), rserial(Z', d, R2'), R2 is R1' + R2'.

From facts:
Z'=c, R1'=20

By rule 1:
rserial(c,d,R2') :- res(c,d,R2')
$\rightarrow$ R2´= 5

R = 35

RULES & RECURSIVITY …. more

Note on recursive rules:

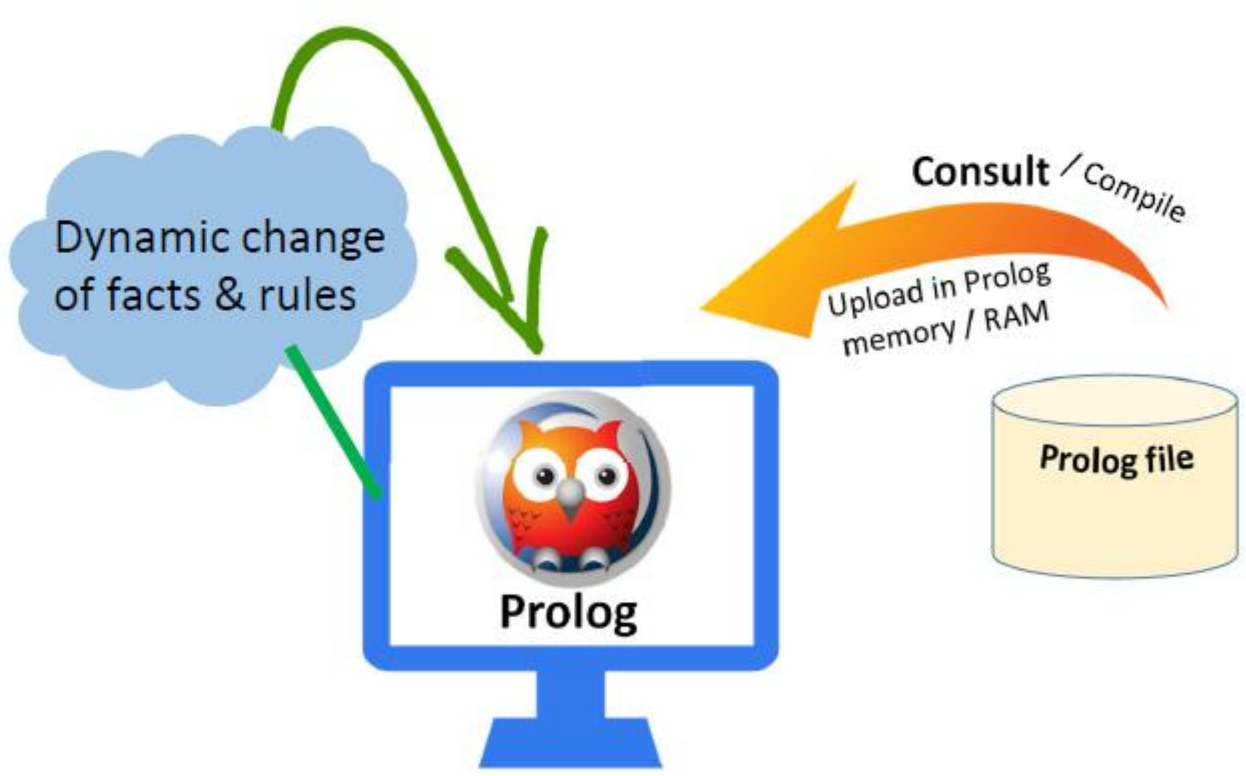
We always start with a "particular" case (the ending case)

```
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.
```

Then, the general case (recursive)

Because …
… when there are 2 (or more) rules with the same conclusion, Prolog always tries first to prove the 1st rule
… if the 1st rule fails, then it tries the 2nd rule

CHANGING THE MEMORY IN PROLOG


Dynamic change of facts & rules
Consult / Compile
Upload in Prolog memory / RAM
Prolog file
Prolog

CHANGING THE MEMORY IN PROLOG

ASSERT

We can dynamically change the memory of Prolog using the following pre-defined predicates:

$$\left.\begin{array}{l}\textbf{assert(S)}\\ \textbf{asserta(S)}\\ \textbf{assertz(S)}\end{array}\right\}\ \textit{To add facts/rules}$$

$$\left.\textbf{retract(S)}\right\}\ \textit{To delete facts/rules}$$

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)

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(luis,ana)).
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

ASSERT

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*

CHANGING THE MEMORY IN PROLOG

ASSERT

```
:- 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.
```

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

|: father(carlos,pedro).

|: finish.
 => Invalid data
|: assert(mother(maria, carlos)).
 => Invalid data
|: end.

true .
```

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

father(antonio, carlos).
father(carlos, pedro).

true.
```

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.
```

How to avoid this answer?

But the memory is empty:

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

true.
```

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.
```

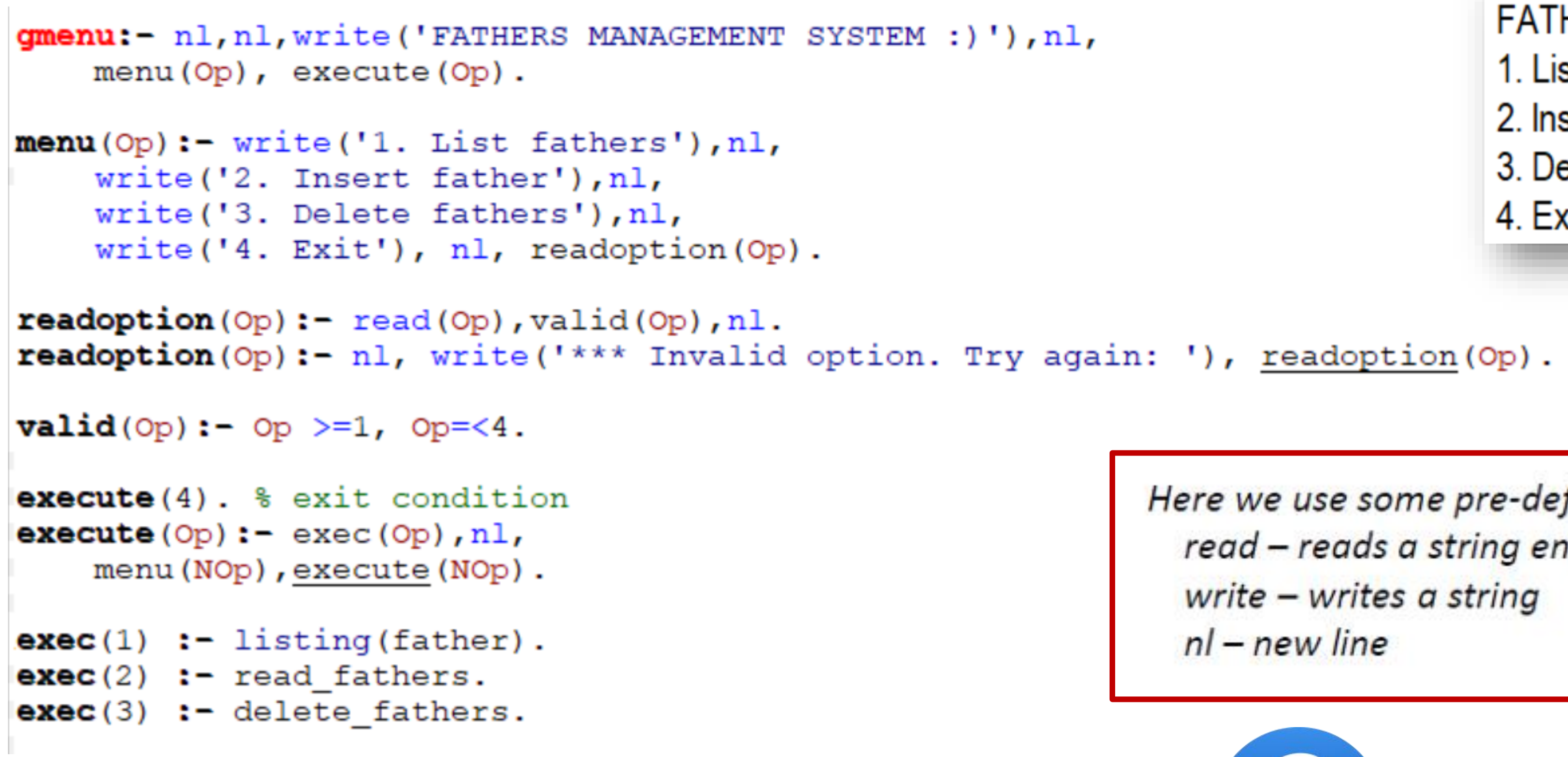
The 2nd rule is only executed when retract fails (i.e. there are no more "fathers" to delete

The 2nd rule simply succeeds

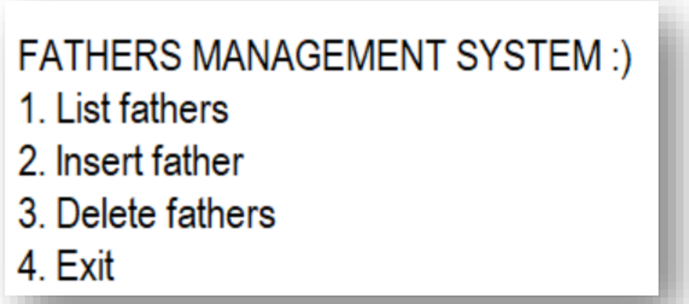
INPUT / OUTPUT

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.
```

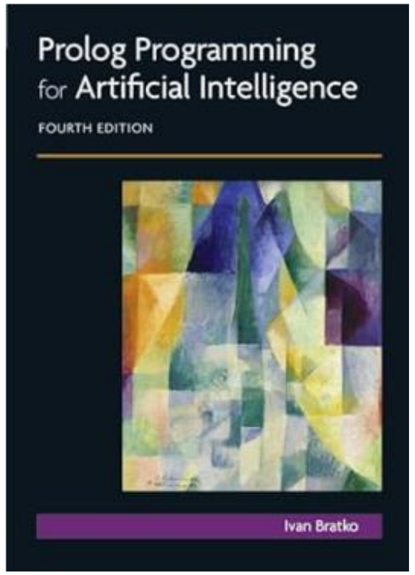


```
FATHERS MANAGEMENT SYSTEM :)
1. List fathers
2. Insert father
3. Delete fathers
4. Exit
```

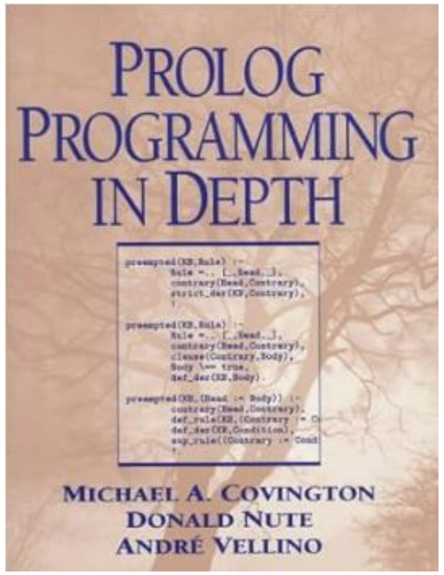
Here we use some pre-defined rules of SWI-Prolog:
- *read – reads a string ended by "."*
- *write – writes a string*
- *nl – new line*

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

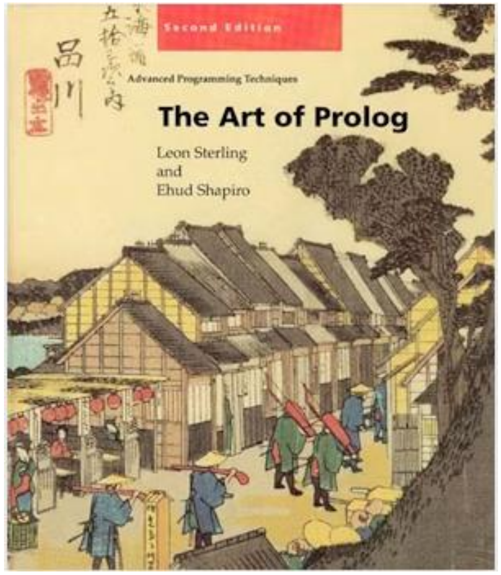
Further reading



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


Second Edition
Advanced Programming Techniques
The Art of Prolog
Leon Sterling and Ehud Shapiro

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