DATA MODELLING IN ENGINEERING

MODELING BASED ON LOGIC PROGRAMMING – PART I -

Ana Inês Oliveira aio@fct.unl.pt

2024 - 2025

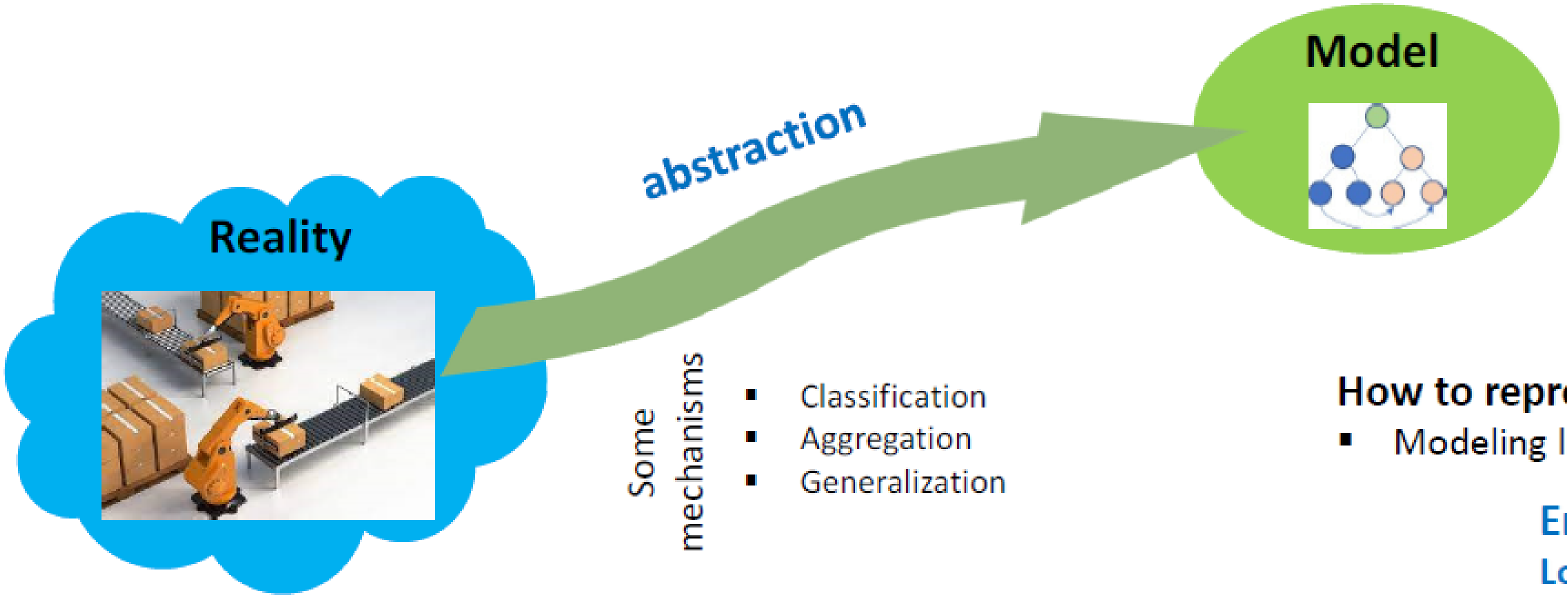
Contents

- **Base Concepts**
  - Modeling
  - Introduction to the PROLOG Language – History
  - PROLOG – Various Implementations
  - SWI PROLOG

- **PROLOG**
  - Representation of Facts
  - Unification
  - Representation of Rules
  - Representation of Queries
  - Backtracking mechanism
  - Recursion mechanism

Modeling

"A model is an **abstract representation** of an environment, system, or entity in the physical, social, or logical world."



What to include?

- Which problem are we interested in?
- Which questions do we want to answer?

Some mechanisms

- Classification
- Aggregation
- Generalization

How to represent?

- Modeling languages

Entity-relationships => DBMS
Logic => PROLOG
"Frames"/Objects/classes => Golog, UML
Knowledge Graph

Modeling is an art

The "quality" of a model depends on our "artistic" skills and experience

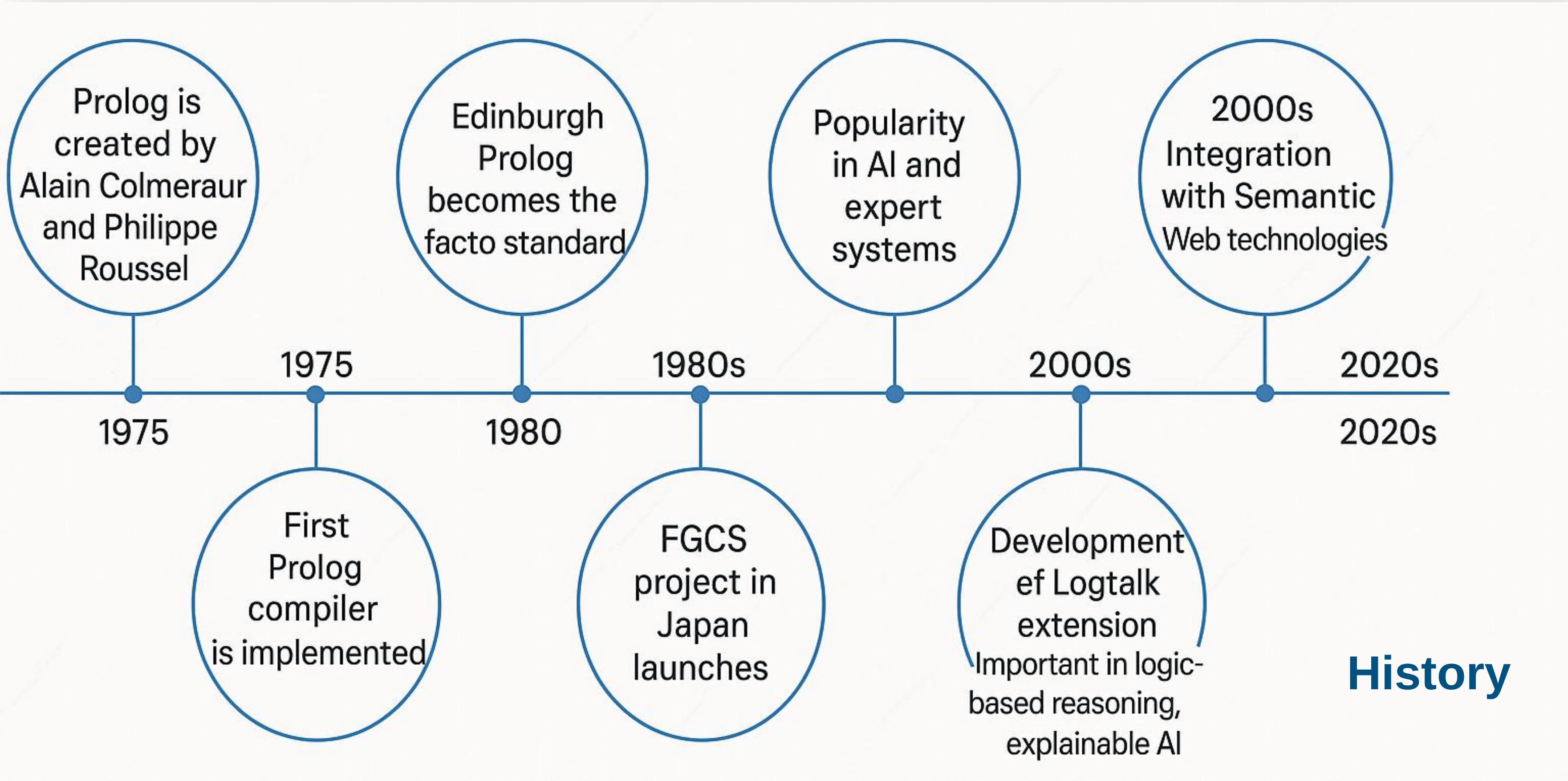
One "metric": How easy is it to answer our questions?

© L.M. Camarinha-Matos 2021-2023

Introduction to the PROLOG Language

Prolog is a logic programming language with an important role in AI.

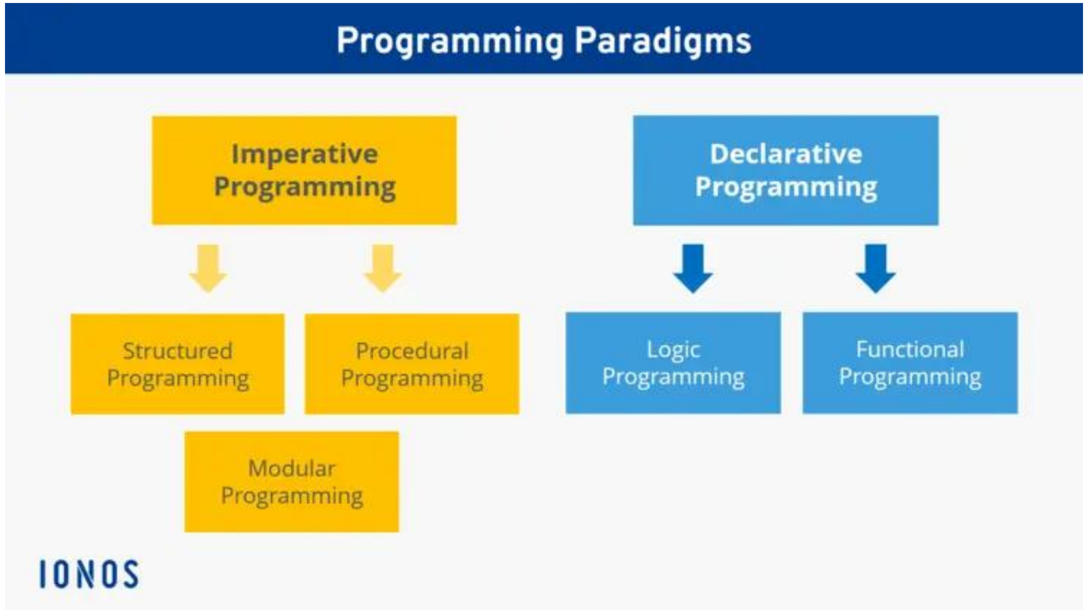
Unlike many other programming languages, Prolog is intended primarily as a **declarative** programming language.


Prolog is created by Alain Colmeraur and Philippe Roussel
1975
1975
First Prolog compiler is implemented
Edinburgh Prolog becomes the facto standard
1980
1980s
FGCS project in Japan launches
Popularity in AI and expert systems
2000s
Development ef Logtalk extension Important in logic-based reasoning, explainable AI
2000s Integration with Semantic Web technologies
2020s
2020s
History

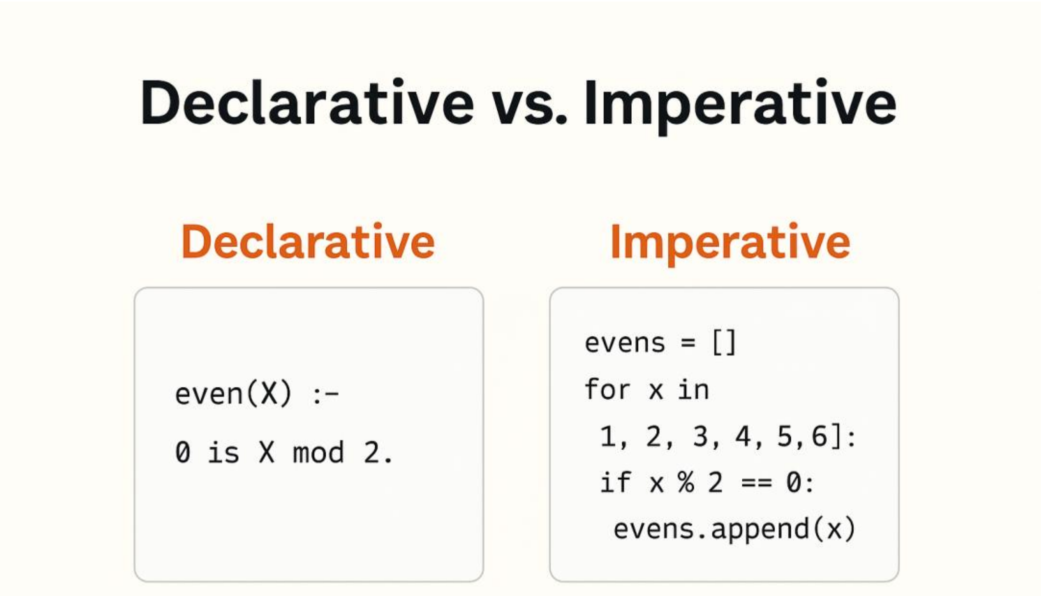
PROLOG: PROgramming in **LOG**ic

In prolog, logic is expressed as **relations** (called as **Facts** and **Rules**).

Introduction to the PROLOG Language



https://www.ionos.co.uk/digitalguide/websites/web-development/declarative-programming/



Declarative vs. Imperative

Declarative

```
even(X) :-
0 is X mod 2.
```

Imperative

```
evens = []
for x in
 1, 2, 3, 4, 5,6]:
 if x % 2 == 0:
  evens.append(x)
```

define what you want the program to accomplish, rather than **how** to do it.

define how the program should do it step by step

PROLOG – Areas for Application

Artificial Intelligence (AI): tasks such as natural language processing, expert systems, automated reasoning, and knowledge representation. Its logical inference capabilities make it well-suited for building intelligent systems.

Expert Systems: is used to develop expert systems, which are computer programs that emulate the decision-making ability of a human expert in a specific domain. These systems use rules and facts encoded in Prolog to provide advice or solutions to complex problems.

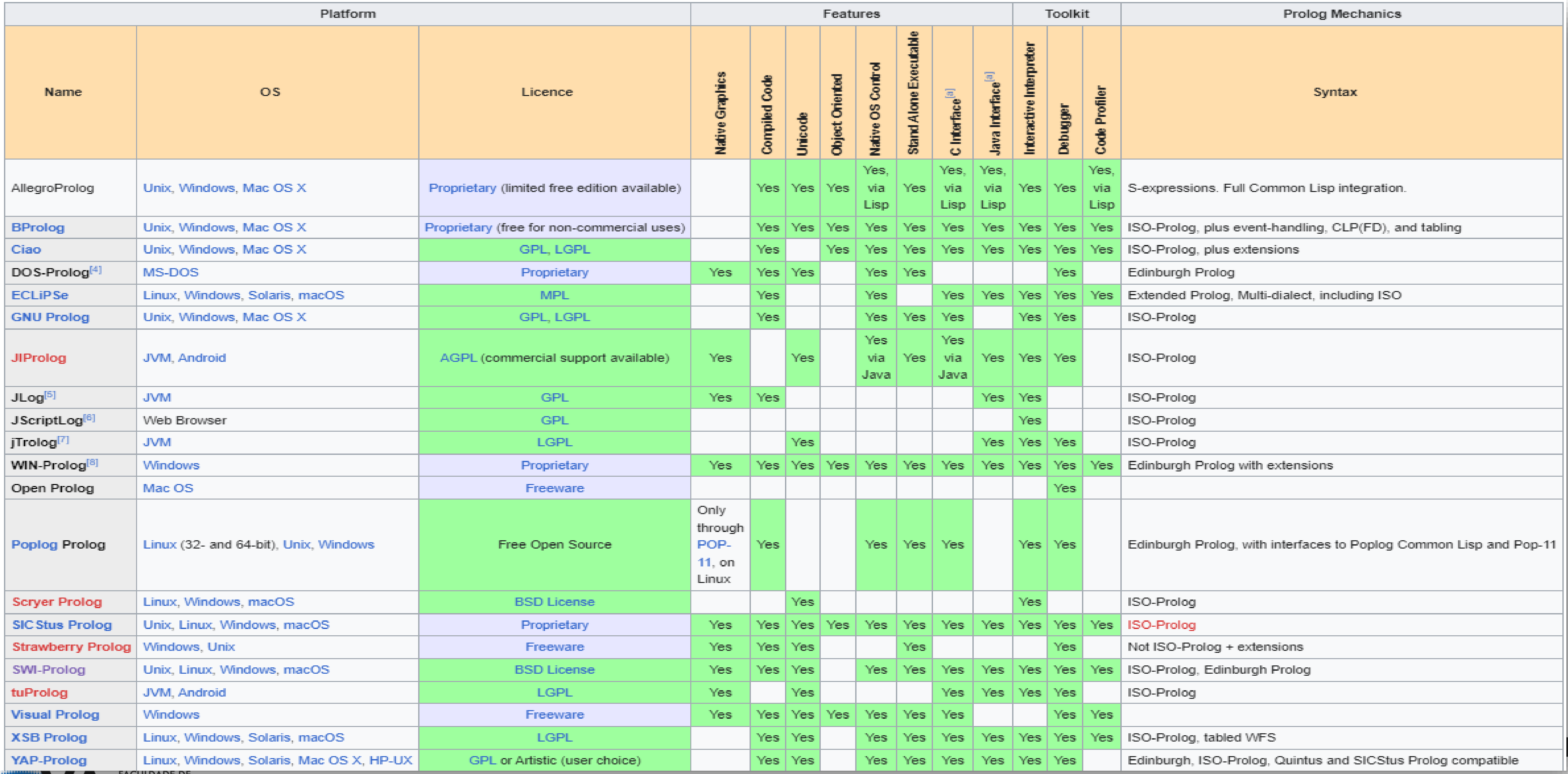
Natural Language Processing (NLP): its pattern-matching capabilities make it suitable for processing and analyzing natural language. It is used in applications such as text parsing, semantic analysis, machine translation, and information retrieval.

Database Systems: can be used to implement database systems, especially in scenarios where complex queries and rule-based reasoning are required. It allows users to express queries and manipulate data using logical predicates.

Symbolic Mathematics: employed in symbolic mathematics for tasks such as theorem proving, symbolic integration, differentiation, and simplification of mathematical expressions. It provides a flexible framework for symbolic computation.

(…)

PROLOG – Various implementations



Platform			Features								Toolkit			Prolog Mechanics
Name	OS	Licence	Native Graphics	Compiled Code	Unicode	Object Oriented	Native OS Control	Stand Alone Executable	C Interface[a]	Java Interface[a]	Interactive Interpreter	Debugger	Code Profiler	Syntax
AllegroProlog	Unix, Windows, Mac OS X	Proprietary (limited free edition available)		Yes	Yes	Yes	Yes, via Lisp	Yes	Yes, via Lisp	Yes, via Lisp	Yes	Yes	Yes, via Lisp	S-expressions. Full Common Lisp integration.
BProlog	Unix, Windows, Mac OS X	Proprietary (free for non-commercial uses)		Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	ISO-Prolog, plus event-handling, CLP(FD), and tabling
Ciao	Unix, Windows, Mac OS X	GPL, LGPL		Yes		Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	ISO-Prolog, plus extensions
DOS-Prolog[4]	MS-DOS	Proprietary	Yes	Yes	Yes		Yes	Yes				Yes		Edinburgh Prolog
ECLiPSe	Linux, Windows, Solaris, macOS	MPL		Yes			Yes		Yes	Yes	Yes	Yes	Yes	Extended Prolog, Multi-dialect, including ISO
GNU Prolog	Unix, Windows, Mac OS X	GPL, LGPL		Yes			Yes	Yes	Yes		Yes	Yes		ISO-Prolog
JIProlog	JVM, Android	AGPL (commercial support available)	Yes		Yes		Yes via Java	Yes	Yes via Java	Yes	Yes	Yes		ISO-Prolog
JLog[5]	JVM	GPL	Yes	Yes						Yes	Yes			ISO-Prolog
JScriptLog[6]	Web Browser	GPL									Yes			ISO-Prolog
jTrolog[7]	JVM	LGPL			Yes					Yes	Yes	Yes		ISO-Prolog
WIN-Prolog[8]	Windows	Proprietary	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Edinburgh Prolog with extensions
Open Prolog	Mac OS	Freeware										Yes		
Poplog Prolog	Linux (32- and 64-bit), Unix, Windows	Free Open Source	Only through POP-11, on Linux	Yes			Yes	Yes	Yes		Yes	Yes		Edinburgh Prolog, with interfaces to Poplog Common Lisp and Pop-11
Scryer Prolog	Linux, Windows, macOS	BSD License			Yes						Yes			ISO-Prolog
SICStus Prolog	Unix, Linux, Windows, macOS	Proprietary	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	ISO-Prolog
Strawberry Prolog	Windows, Unix	Freeware	Yes	Yes	Yes			Yes				Yes		Not ISO-Prolog + extensions
SWI-Prolog	Unix, Linux, Windows, macOS	BSD License	Yes	Yes	Yes		Yes	Yes	Yes	Yes	Yes	Yes	Yes	ISO-Prolog, Edinburgh Prolog
tuProlog	JVM, Android	LGPL	Yes		Yes				Yes	Yes	Yes	Yes		ISO-Prolog
Visual Prolog	Windows	Freeware	Yes	Yes	Yes	Yes	Yes	Yes	Yes			Yes	Yes	
XSB Prolog	Linux, Windows, Solaris, macOS	LGPL		Yes	Yes		Yes	Yes	Yes	Yes	Yes	Yes	Yes	ISO-Prolog, tabled WFS
YAP-Prolog	Linux, Windows, Solaris, Mac OS X, HP-UX	GPL or Artistic (user choice)		Yes	Yes		Yes	Yes	Yes	Yes	Yes	Yes		Edinburgh, ISO-Prolog, Quintus and SICStus Prolog compatible

https://en.wikipedia.org/wiki/Comparison_of_Prolog_implementations

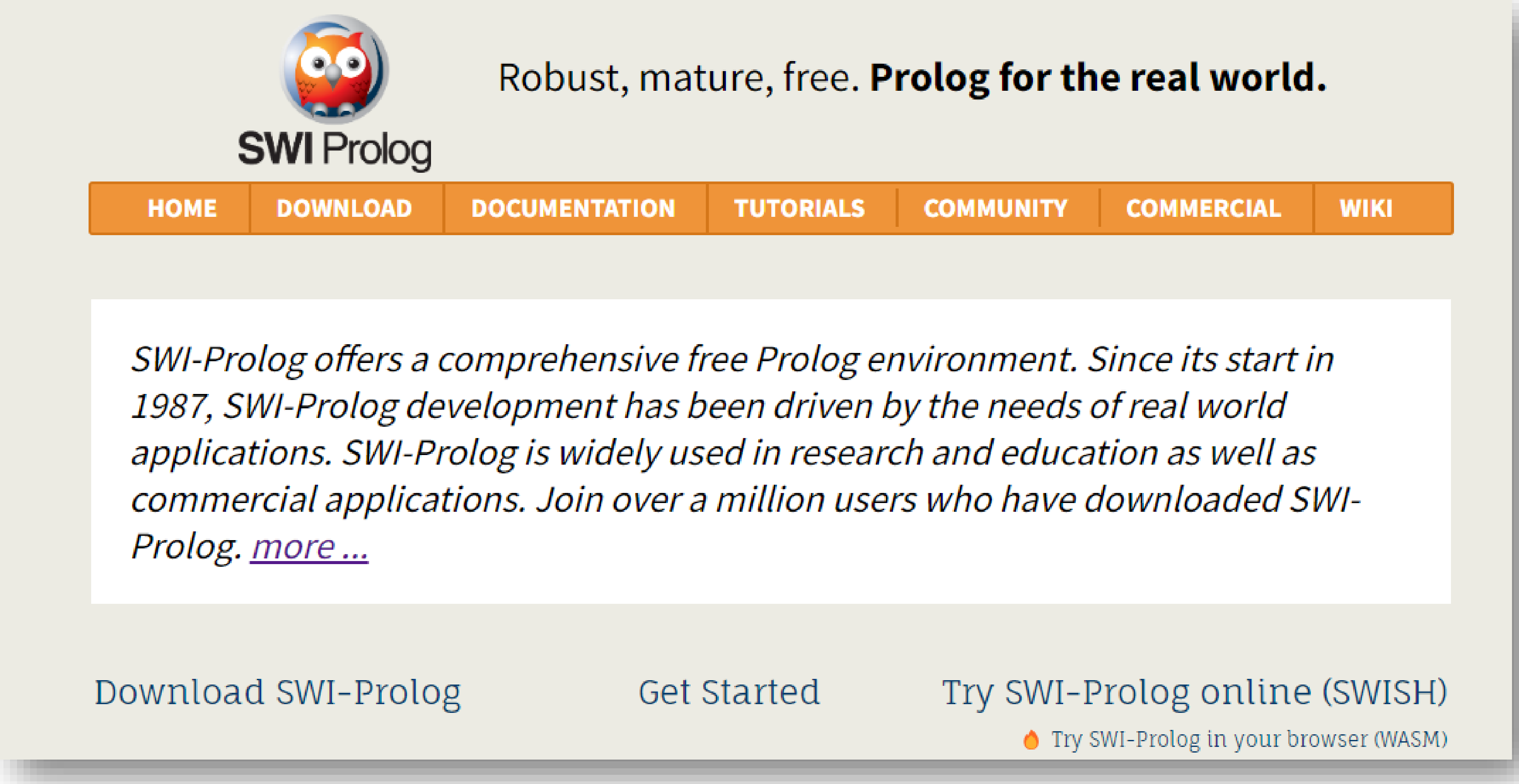
SWI PROLOG

SWI-Prolog is a **free implementation** of the programming language **Prolog**, commonly used for teaching and semantic web application

SWI-Prolog has been under **continuous development** since 1987. Its main author is Jan Wielemaker. The name SWI is derived from Sociaal-Wetenschappelijke Informatica ("Social Science Informatics"), the former name of the group at the University of Amsterdam, where Wielemaker was employed when he initiated the development of SWI-Prolog.

https://en.wikipedia.org/wiki/SWI-Prolog

https://www.swi-prolog.org/



Robust, mature, free. **Prolog for the real world.**

SWI-Prolog offers a comprehensive free Prolog environment. Since its start in 1987, SWI-Prolog development has been driven by the needs of real world applications. SWI-Prolog is widely used in research and education as well as commercial applications. Join over a million users who have downloaded SWI-Prolog. more ...

Download SWI-Prolog

Get Started

Try SWI-Prolog online (SWISH)

Try SWI-Prolog in your browser (WASM)

PROLOG – Representation of Facts

PROLOG is a declarative programming language, meaning that it allows the programmer to specify the **rules** and **facts** about a problem domain, and then the Prolog interpreter will use these rules and facts to automatically infer solutions to problems.

Facts:

Statements about what is **true** in our modeling world

Example 1:

Fact:		*One possible representation in Prolog:*
board is black	→	black(board).
table is brown	→	brown(table).
chair is brown	→	brown(chair).
table is made of wood	→	made_of(table, wood).

facts

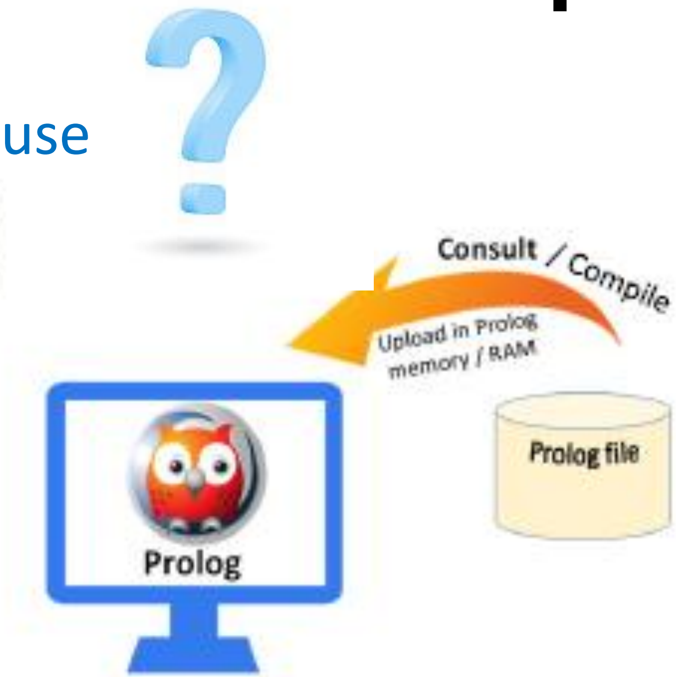
Anything not explicitly stated is considered **false**

Adapted from © Slide from Prof. L.M. Camarinha-Matos

PROLOG – Representation of Facts

How to use

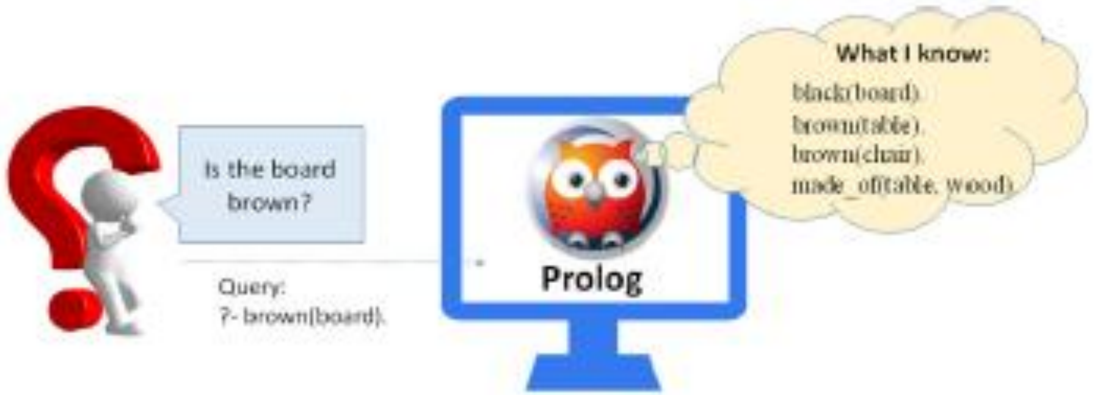
1



Edit a text file of Facts → xxxx.pl
Then upload it in the Prolog memory.

Prolog requires the facts to be in main memory (RAM).

2 Ask questions → **Queries**



?- black(board).
true
?-brown(board).
false
?- brown(X).
X=(table)

constant

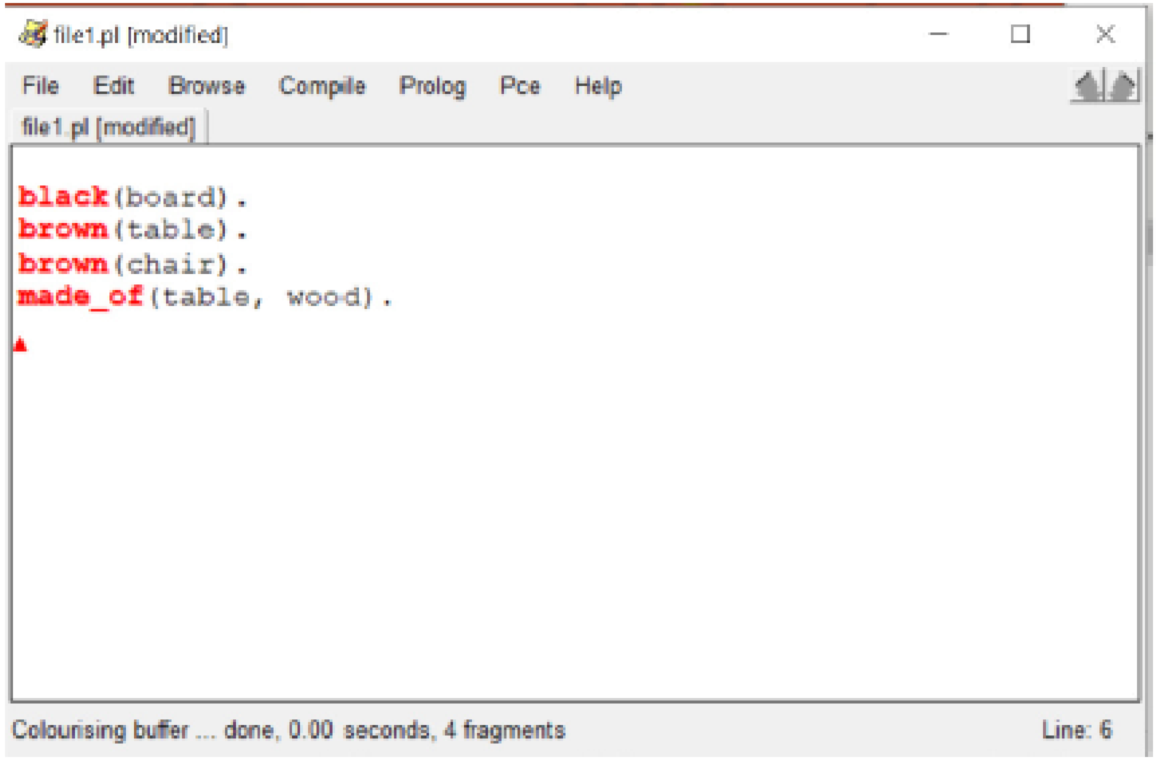
variable

Adapted form © Slide from Prof. L.M. Camarinha-Matos

PROLOG – Representation of Facts

Using SWI-PROLOG

Text editor window:



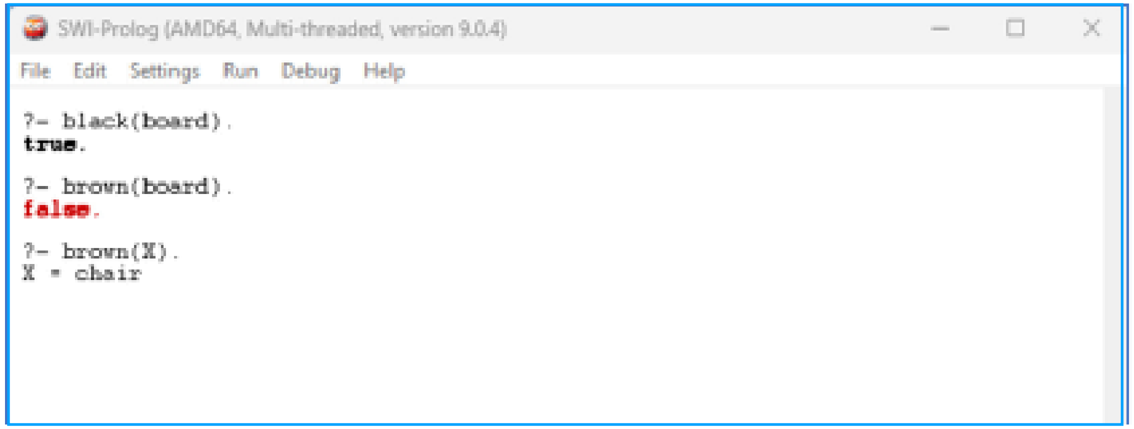
```
black(board).
brown(table).
brown(chair).
made_of(table, wood).
```

Colourising buffer ... done, 0.00 seconds, 4 fragments

Line: 6

Compile buffer
(or File -> Consult)

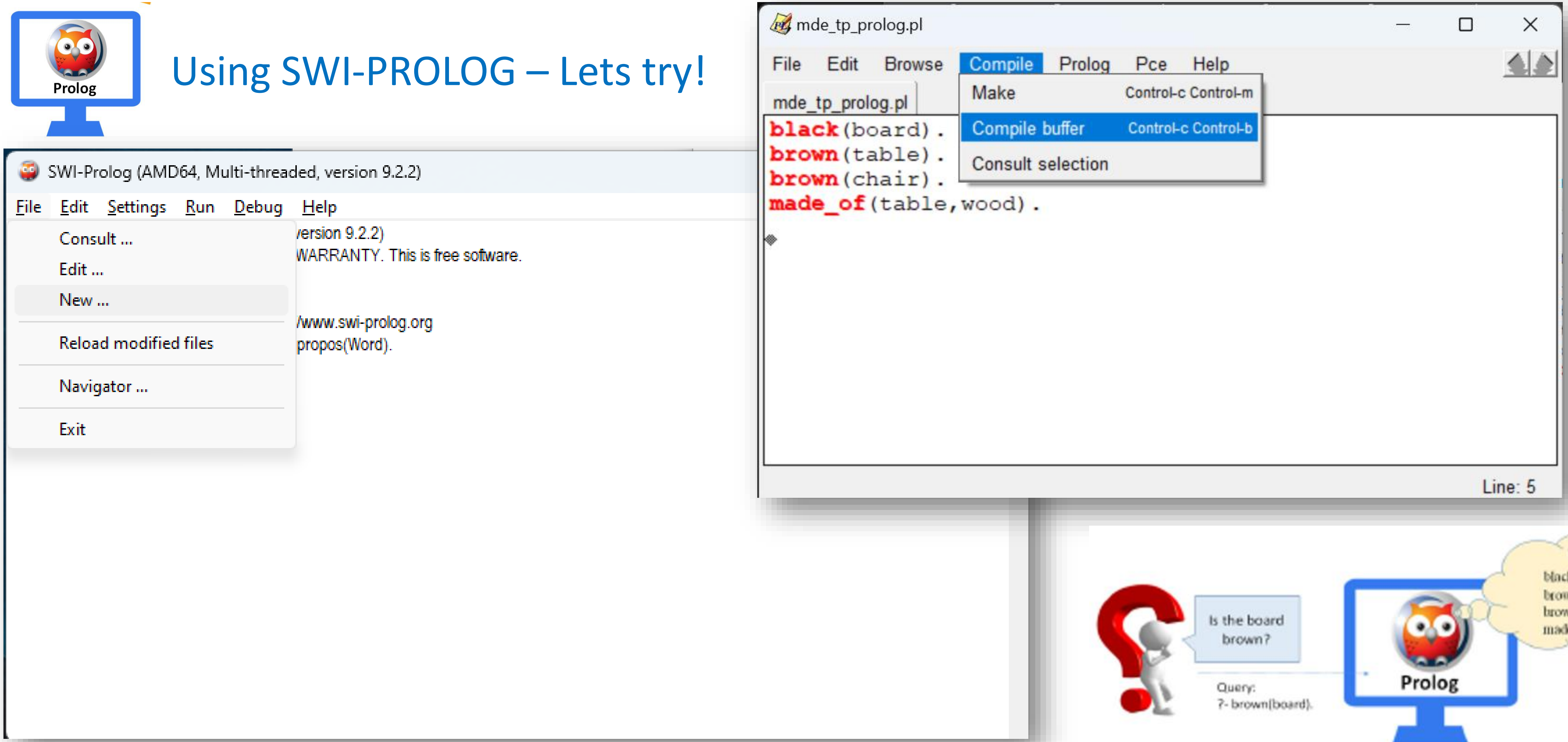
Prolog query window:


SWI-Prolog (AMD64, Multi-threaded, version 9.0.4)
File Edit Settings Run Debug Help

```
?- black(board).
true.

?- brown(board).
false.

?- brown(X).
X = chair
```

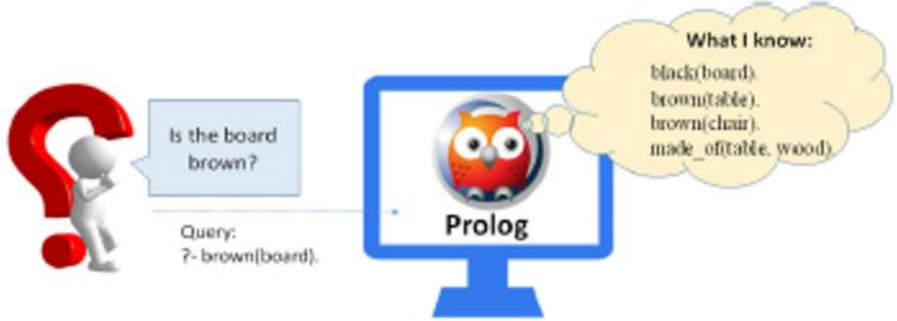
PROLOG – Representation of Facts

Using SWI-PROLOG – Lets try!


SWI-Prolog (AMD64, Multi-threaded, version 9.2.2)
File Edit Settings Run Debug Help
Consult ...
Edit ...
New ...
Reload modified files
Navigator ...
Exit
mde_tp_prolog.pl
File Edit Browse Compile Prolog Pce Help
Make Control-c Control-m
Compile buffer Control-c Control-b
Consult selection
Line: 5

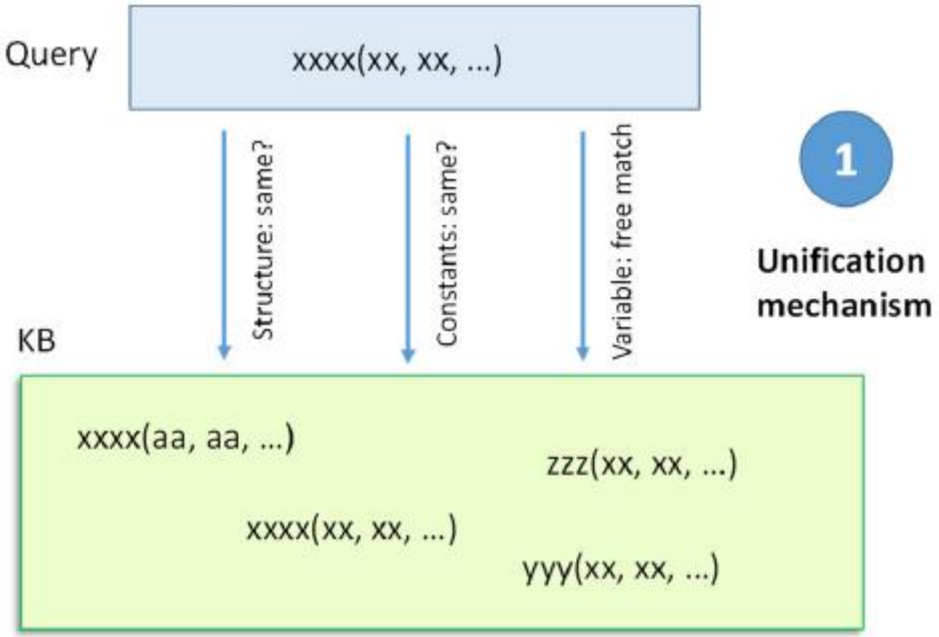
```
version 9.2.2)
WARRANTY. This is free software.

/www.swi-prolog.org
propos(Word).
```

```
black(board).
brown(table).
brown(chair).
made_of(table,wood).
```



PROLOG – UNIFICATION

Unification in Prolog is a fundamental concept that involves matching and aligning terms in logic programming



Unification – the pattern matching mechanism
- constants match with exactly the same constant
- variables can match with everything

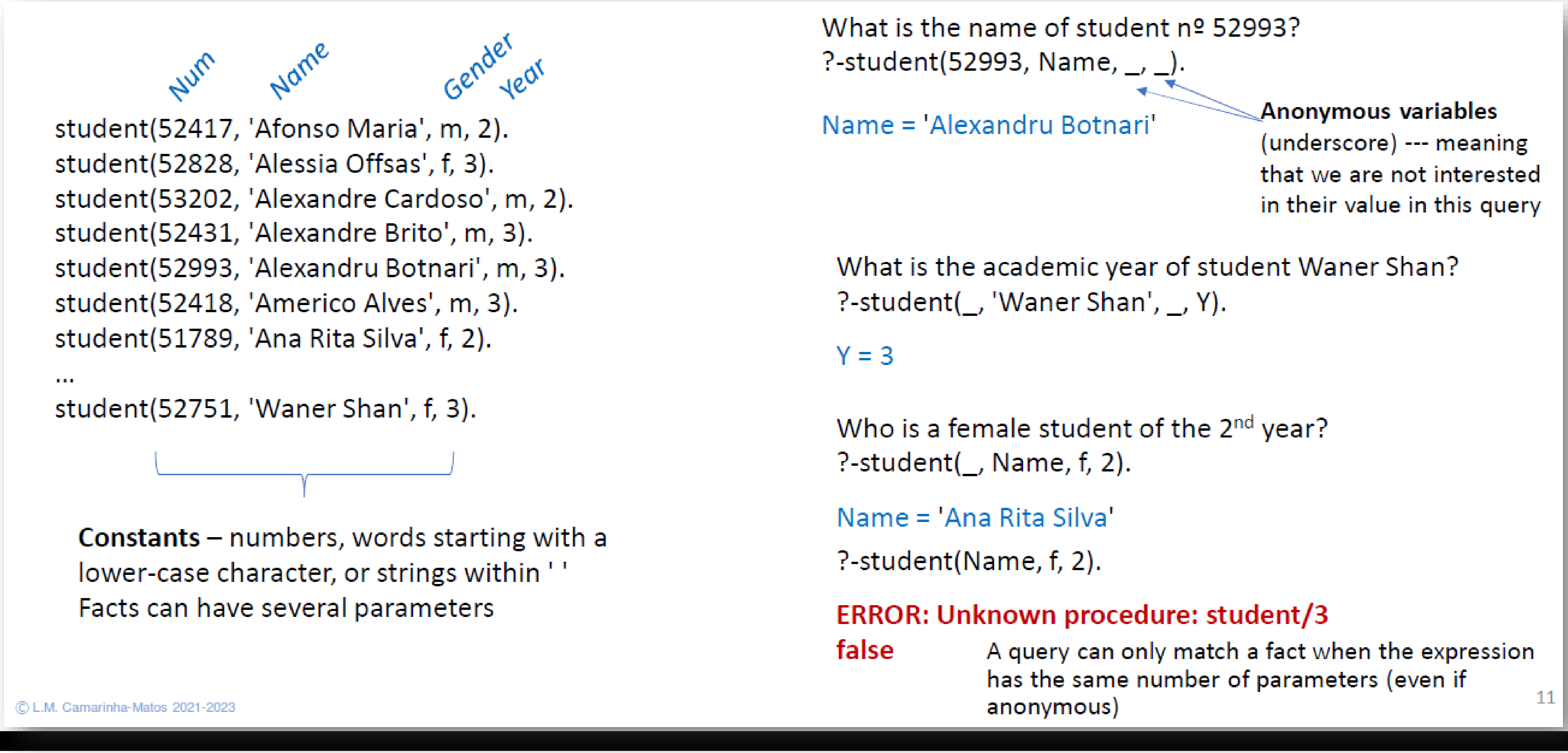
```
father(john, mary).

?-father(X, Y).
X=john,
Y=mary.
```

```
black(board).
brown(table).
brown(chair).
made_of(table,wood).

?-brown(Thing).
Thing = table ;
Thing = chair.
```

Prolog starts giving the first answer;
then if the user enters ";" it gives the 2nd answer …

Adapted form © Slide from Prof. L.M. Camarinha-Matos

PROLOG – UNIFICATION



Num Name Gender Year

```
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).
```

Constants – numbers, words starting with a lower-case character, or strings within ' '
Facts can have several parameters

© L.M. Camarinha-Matos 2021-2023

What is the name of student nº 52993?

```
?-student(52993, Name, _, _).
```

```
Name = 'Alexandru Botnari'
```

Anonymous variables (underscore) --- meaning that we are not interested in their value in this query

What is the academic year of student Waner Shan?

```
?-student(_, 'Waner Shan', _, Y).
```

```
Y = 3
```

Who is a female student of the 2nd year?

```
?-student(_, Name, f, 2).
```

```
Name = 'Ana Rita Silva'
```

```
?-student(Name, f, 2).
```

ERROR: Unknown procedure: student/3

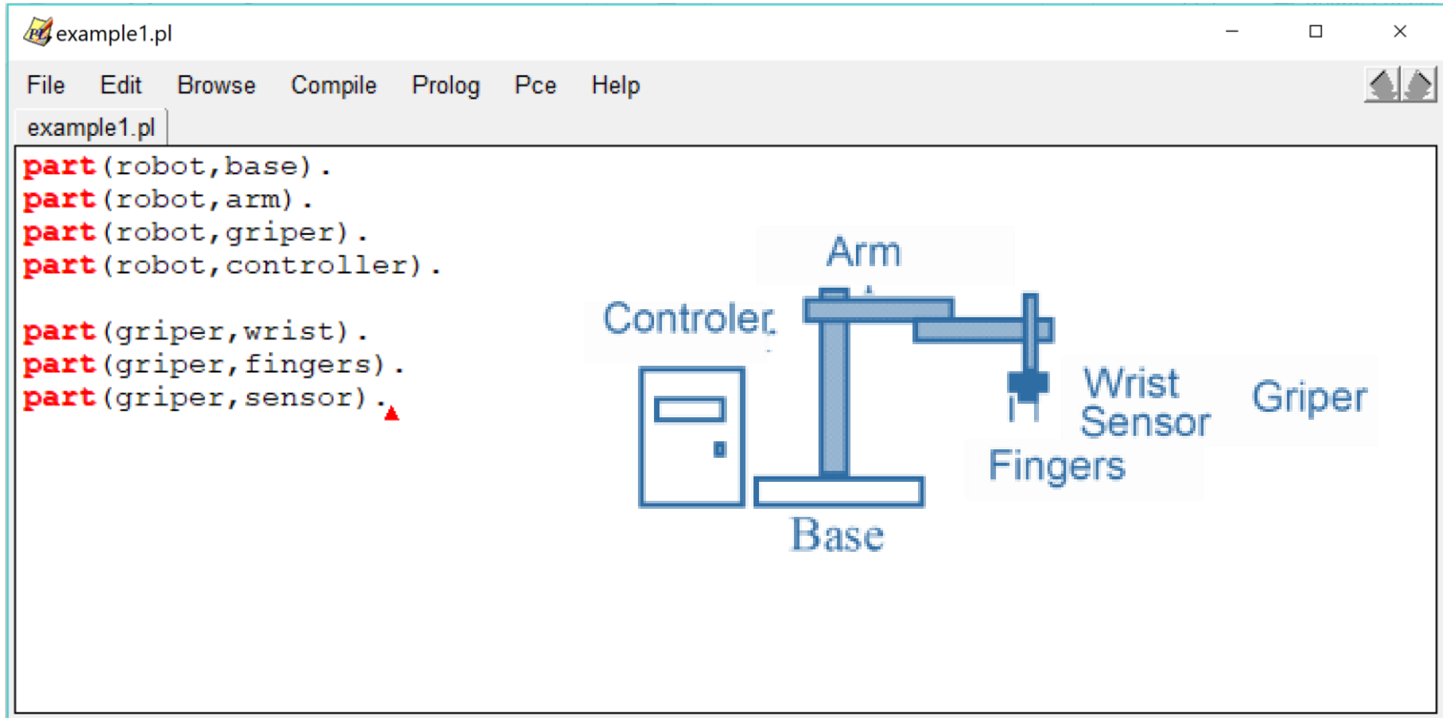
false

A query can only match a fact when the expression has the same number of parameters (even if anonymous)

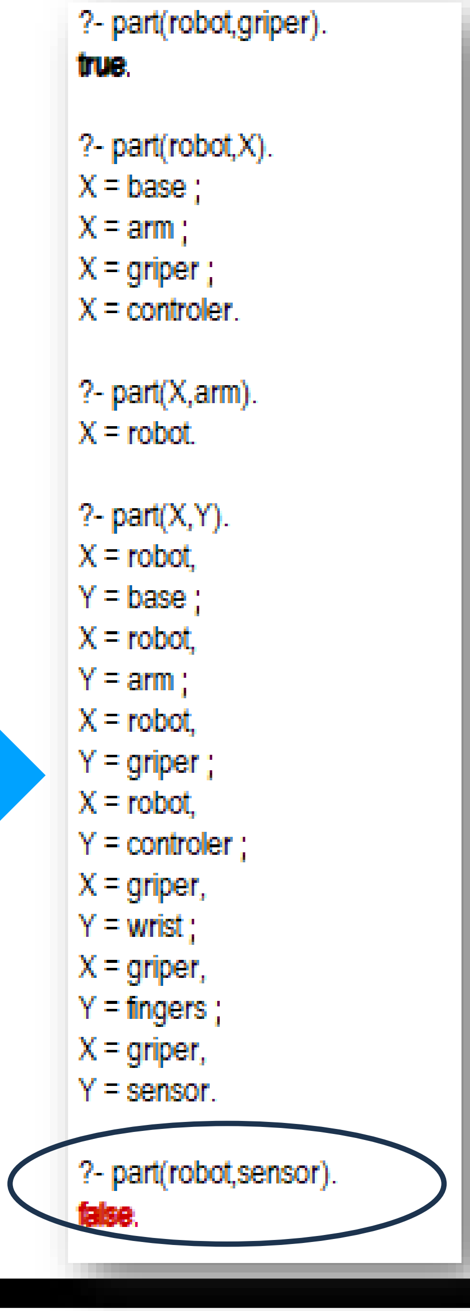
11

PROLOG – Another example

Example 1: Model the structure of a robot

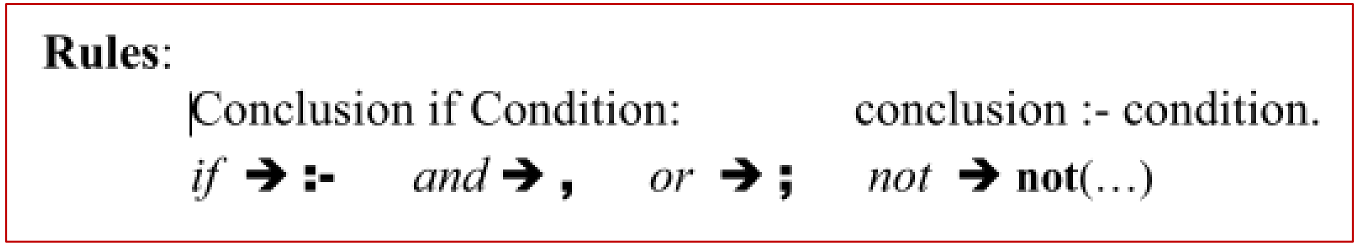


```
part(robot,base).
part(robot,arm).
part(robot,griper).
part(robot,controller).

part(griper,wrist).
part(griper,fingers).
part(griper,sensor).
```



```
?- part(robot,griper).
true.

?- part(robot,X).
X = base ;
X = arm ;
X = griper ;
X = controler.

?- part(X,arm).
X = robot.

?- part(X,Y).
X = robot,
Y = base ;
X = robot,
Y = arm ;
X = robot,
Y = griper ;
X = robot,
Y = controler ;
X = griper,
Y = wrist ;
X = griper,
Y = fingers ;
X = griper,
Y = sensor.

?- part(robot,sensor).
false.
```

PROLOG – Representation of Rules



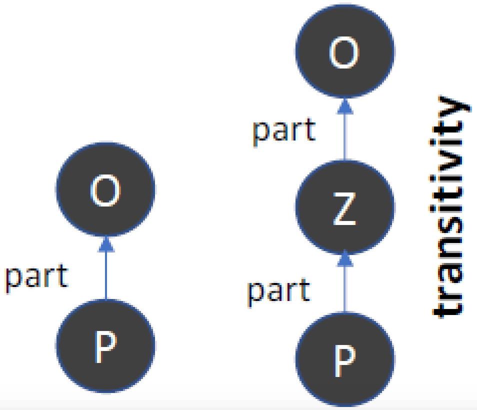
Rules:

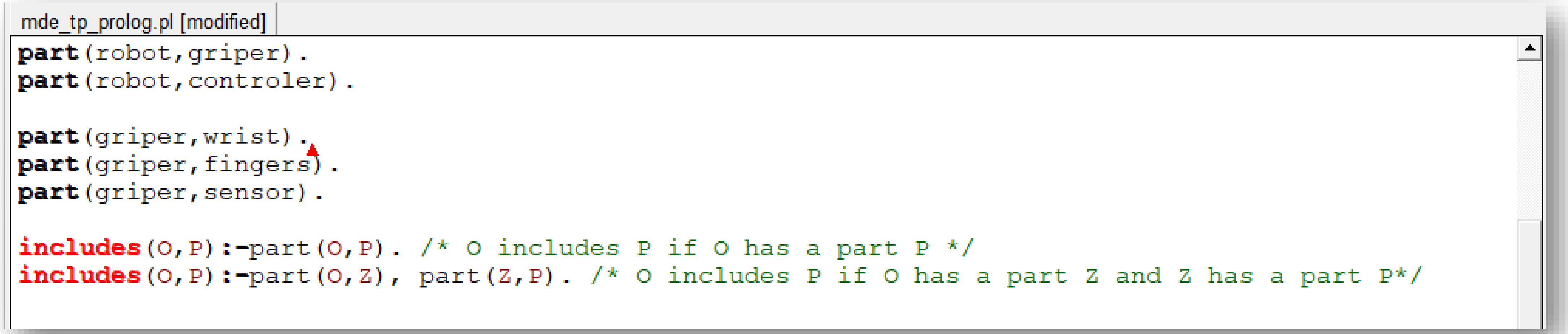
Conclusion if Condition: conclusion :- condition.

if → **:-** *and* → **,** *or* → **;** *not* → **not**(…)

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*/

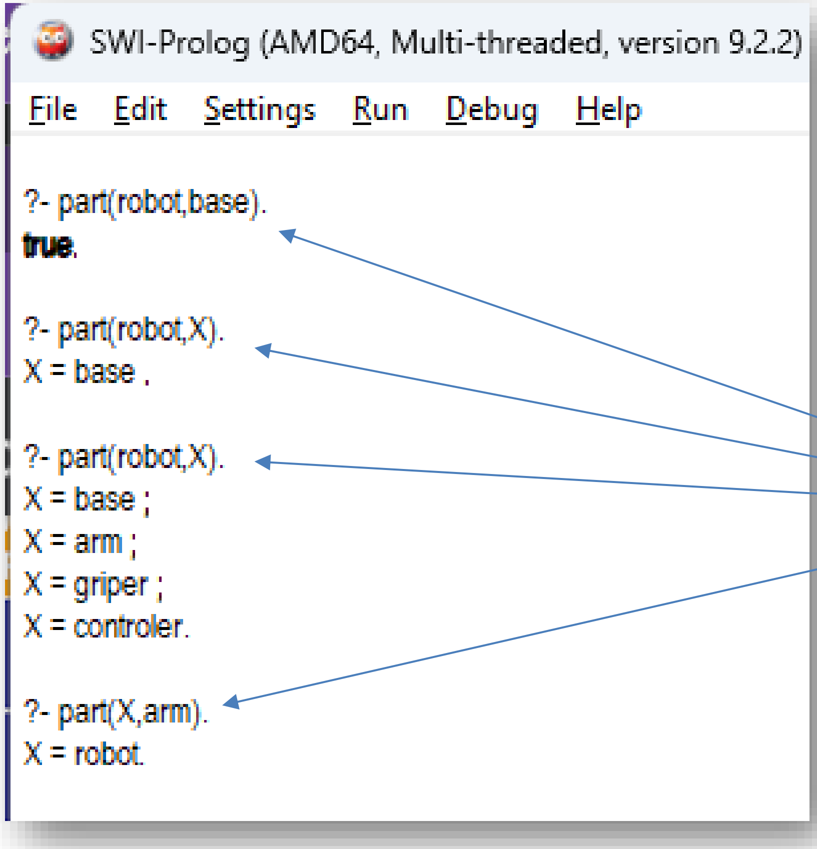




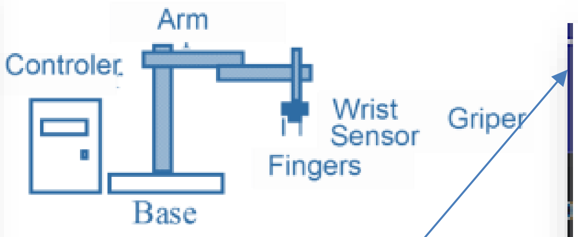
```
part(robot,griper).
part(robot,controler).

part(griper,wrist).
part(griper,fingers).
part(griper,sensor).

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*/
```

SWI Prolog

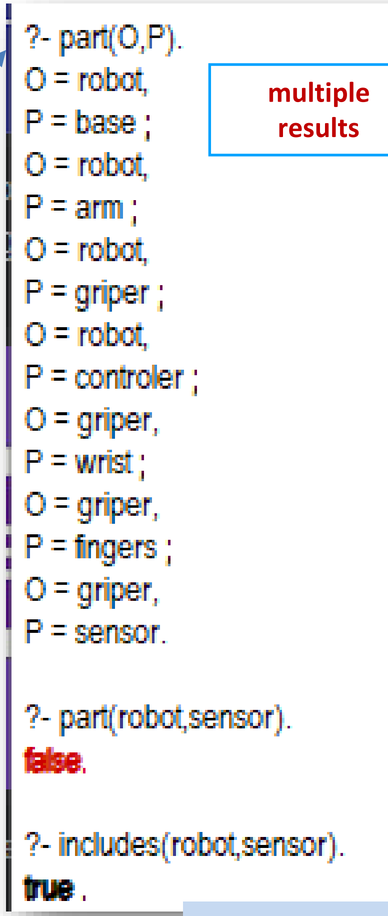
PROLOG – Representation of Queries



```
?- part(robot,base).
true.

?- part(robot,X).
X = base .

?- part(robot,X).
X = base ;
X = arm ;
X = griper ;
X = controler.

?- part(X,arm).
X = robot.
```



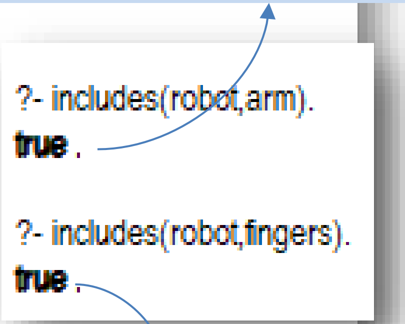
Examples "?-"



```
?- part(O,P).
O = robot,
P = base ;
O = robot,
P = arm ;
O = robot,
P = griper ;
O = robot,
P = controler ;
O = griper,
P = wrist ;
O = griper,
P = fingers ;
O = griper,
P = sensor.

?- part(robot,sensor).
false.

?- includes(robot,sensor).
true .
```

multiple results

Answer obtained from the first rule:
includes(robot, arm) :- part(robot, arm)



```
?- includes(robot,arm).
true .

?- includes(robot,fingers).
true .
```

Answer obtained from the second rule:
includes(robot, fingers) :- part(Z,fingers),part(robot,Z).

SWI Prolog

NOVA FACULDADE DE CIÊNCIAS E TECNOLOGIA

PROLOG – Representation of Queries

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

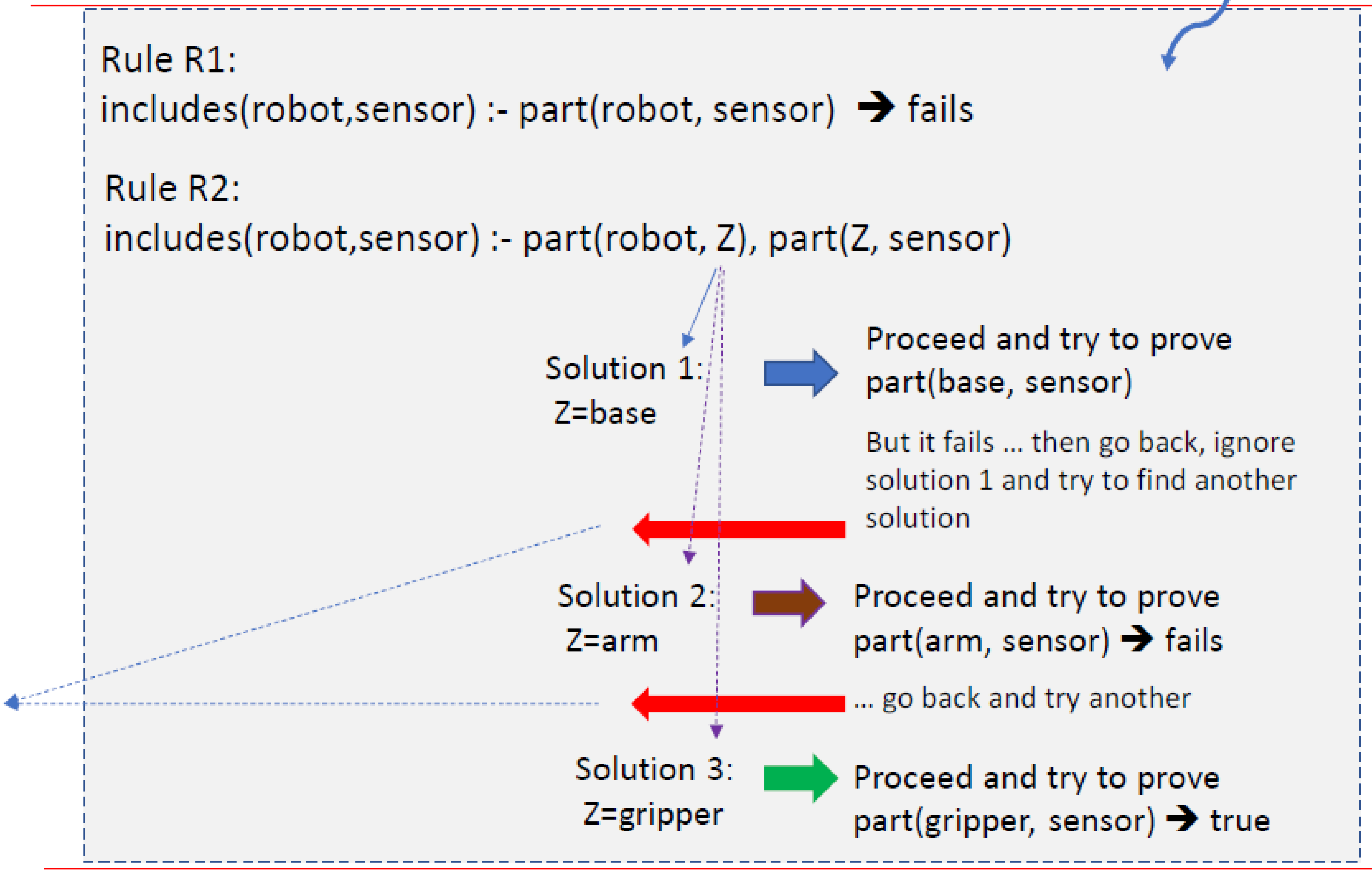
R1 includes(O,P) :- part(O,P).

R2 includes(O, P) :- part(O,Z),
                     part(Z,P).

Backtracking mechanism
Going back and trying to find another solution

?-includes(robot, sensor).

The internal reasoning of Prolog (invisible to the user)



Rule R1:
includes(robot,sensor) :- part(robot, sensor) ➔ fails

Rule R2:
includes(robot,sensor) :- part(robot, Z), part(Z, sensor)

Solution 1:
Z=base

Proceed and try to prove part(base, sensor)

But it fails ... then go back, ignore solution 1 and try to find another solution

Solution 2:
Z=arm

Proceed and try to prove part(arm, sensor) ➔ fails

... go back and try another

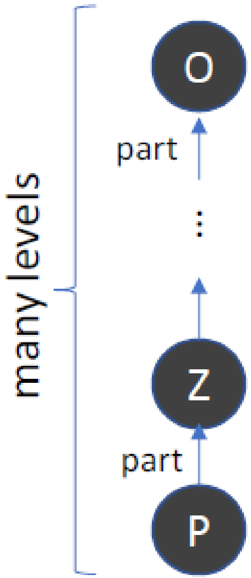
Solution 3:
Z=gripper

Proceed and try to prove part(gripper, sensor) ➔ true

true

PROLOG – Representation of Queries

We can generalize the rule



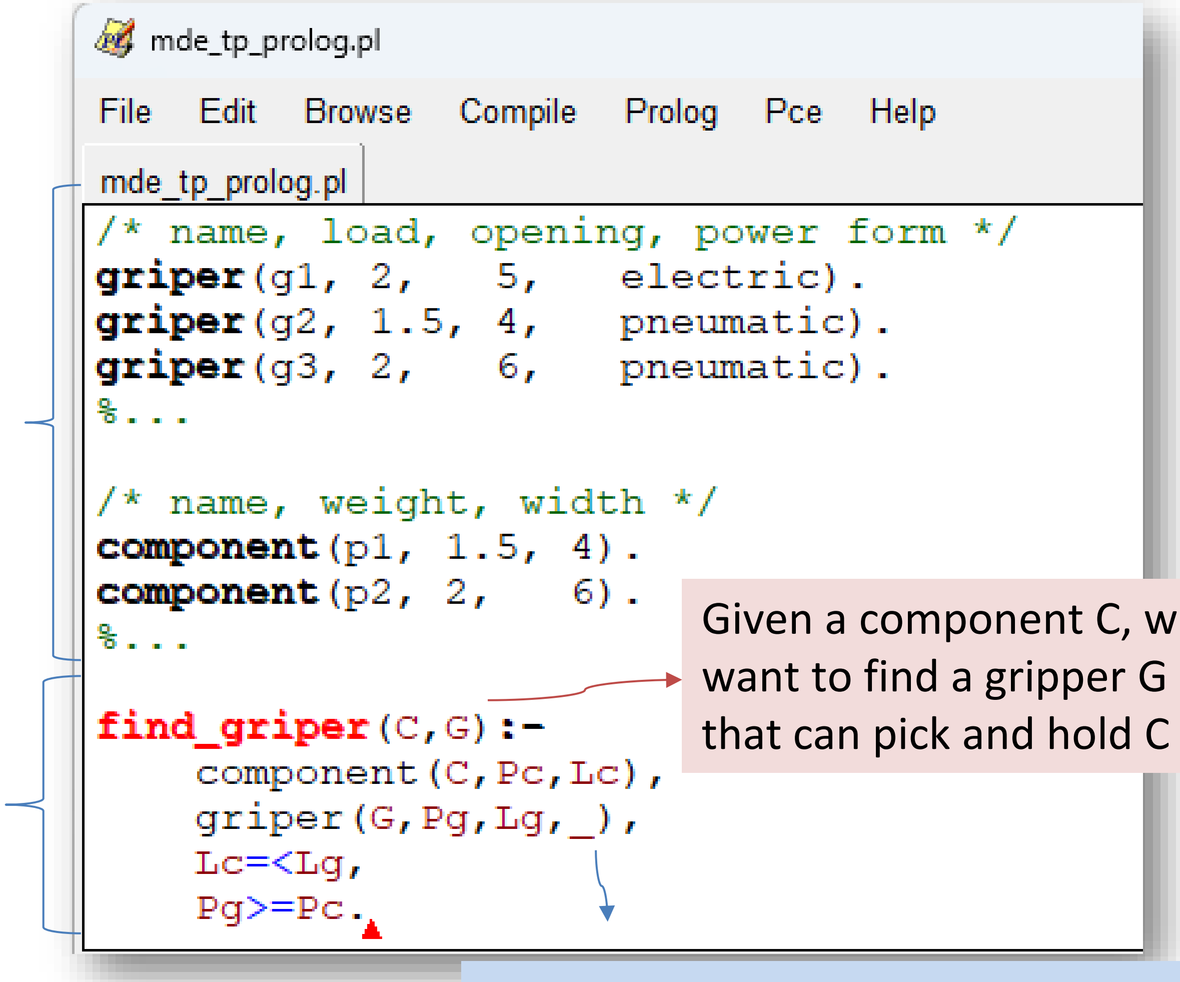
```
contains(O,P) :- part(O,P).                 /* O contains P if O has part P */
contains(O,P) :- part(Z,P), contains(O,Z).  /* O contains P if Z has part P and O contains Z */
```

The 2nd rule is defined in terms of itself
… i.e., recursive definition

recursion mechanism

PROLOG – Another example

Example 2: Robot components



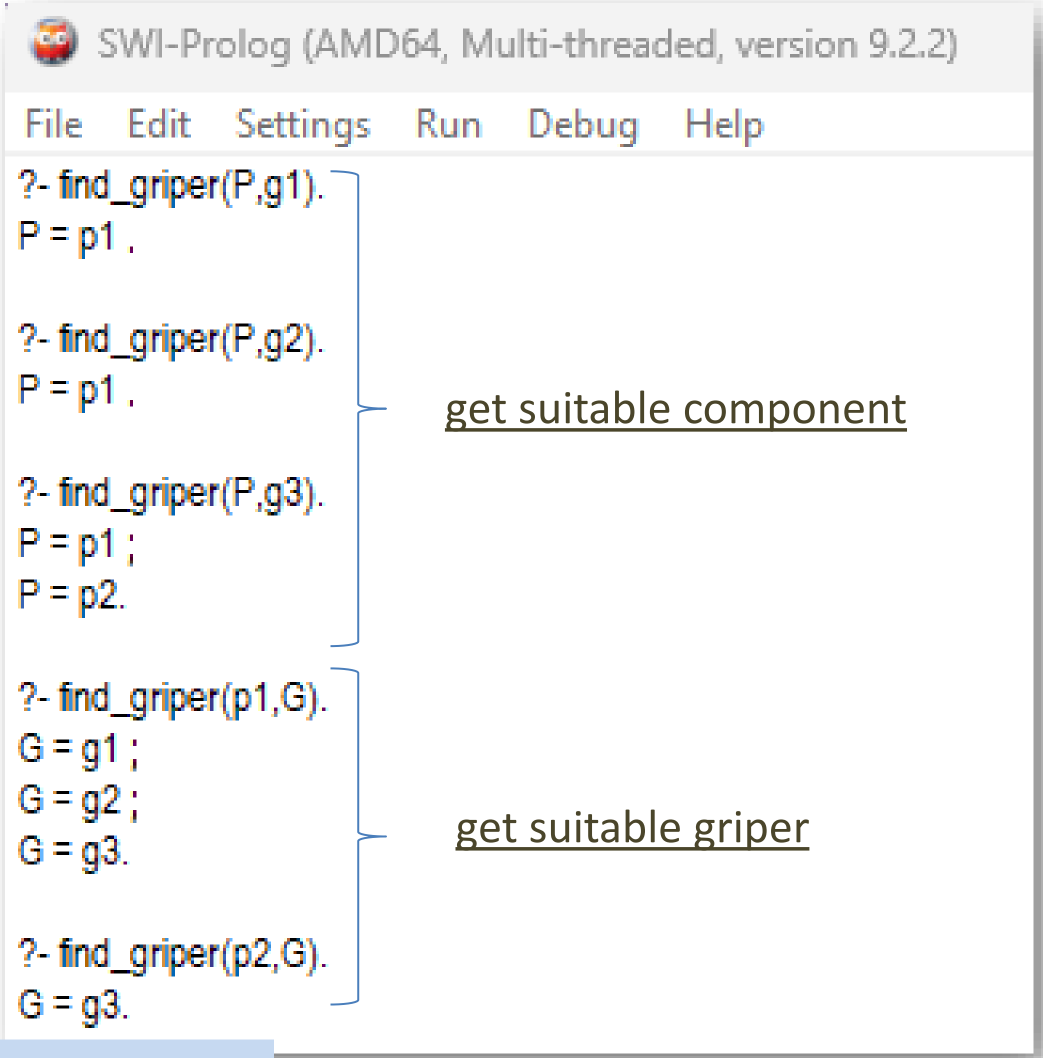
Facts

```
/* name, load, opening, power form */
griper(g1, 2,   5,    electric).
griper(g2, 1.5, 4,    pneumatic).
griper(g3, 2,   6,    pneumatic).
%...

/* name, weight, width */
component(p1, 1.5, 4).
component(p2, 2,   6).
%...
```

Rule

```
find_griper(C,G):-
    component(C,Pc,Lc),
    griper(G,Pg,Lg,_),
    Lc=<Lg,
    Pg>=Pc.
```

Given a component C, we want to find a gripper G that can pick and hold C


SWI-Prolog (AMD64, Multi-threaded, version 9.2.2)
File Edit Settings Run Debug Help

```
?- find_griper(P,g1).
P = p1 .

?- find_griper(P,g2).
P = p1 .

?- find_griper(P,g3).
P = p1 ;
P = p2.
```

get suitable component

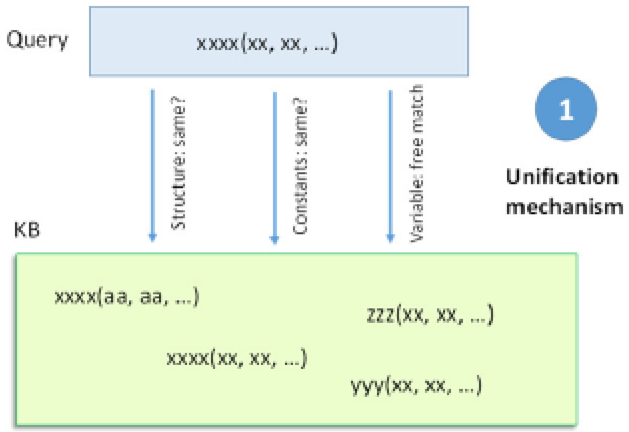
```
?- find_griper(p1,G).
G = g1 ;
G = g2 ;
G = g3.

?- find_griper(p2,G).
G = g3.
```

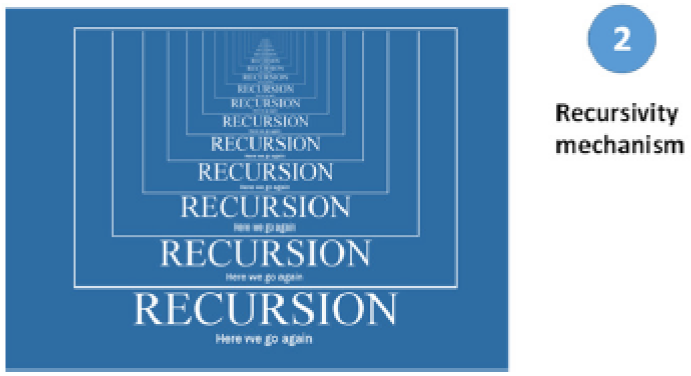
get suitable griper

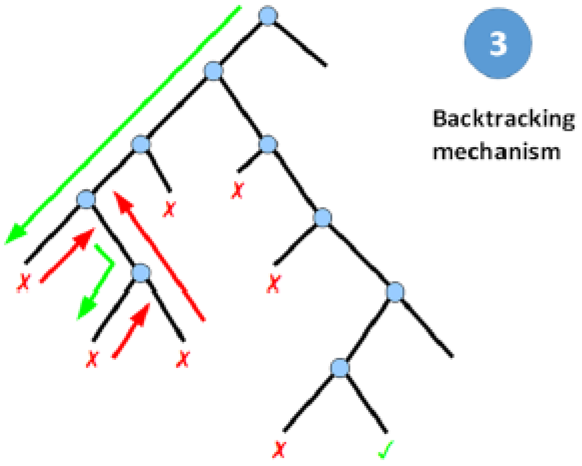
Anonymous variable (represented by underscore) … meaning that we are not interested in it (for this rule)

PROLOG – 3 Base Mechanisms

In summary:







Unification

Matching query terms with facts/rules

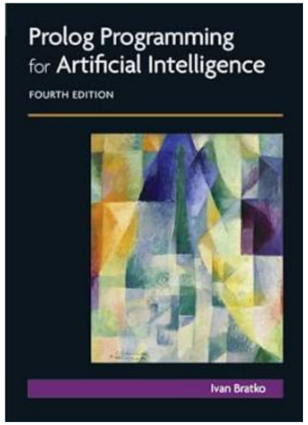
Recursion

Defining concepts using themselves

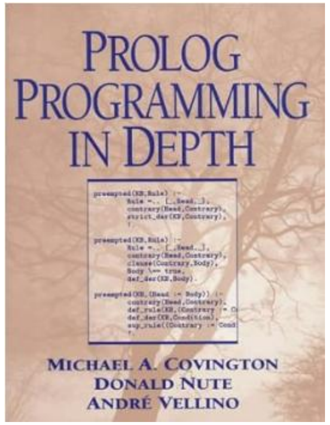
Backtracking

Exploring alternatives when a match fails

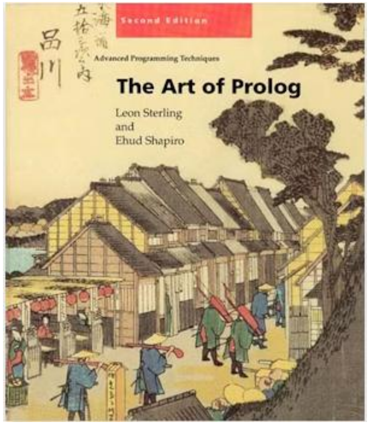
Further reading



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/

(…)

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

WIKIBOOKS Open books for an open world 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