NOVA SCHOOL OF SCIENCE & TECHNOLOGY

MDE

Modeling with PROLOG
- Part I -

Ana Inês Oliveira

Nova University of Lisbon
School of Science and Technology

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

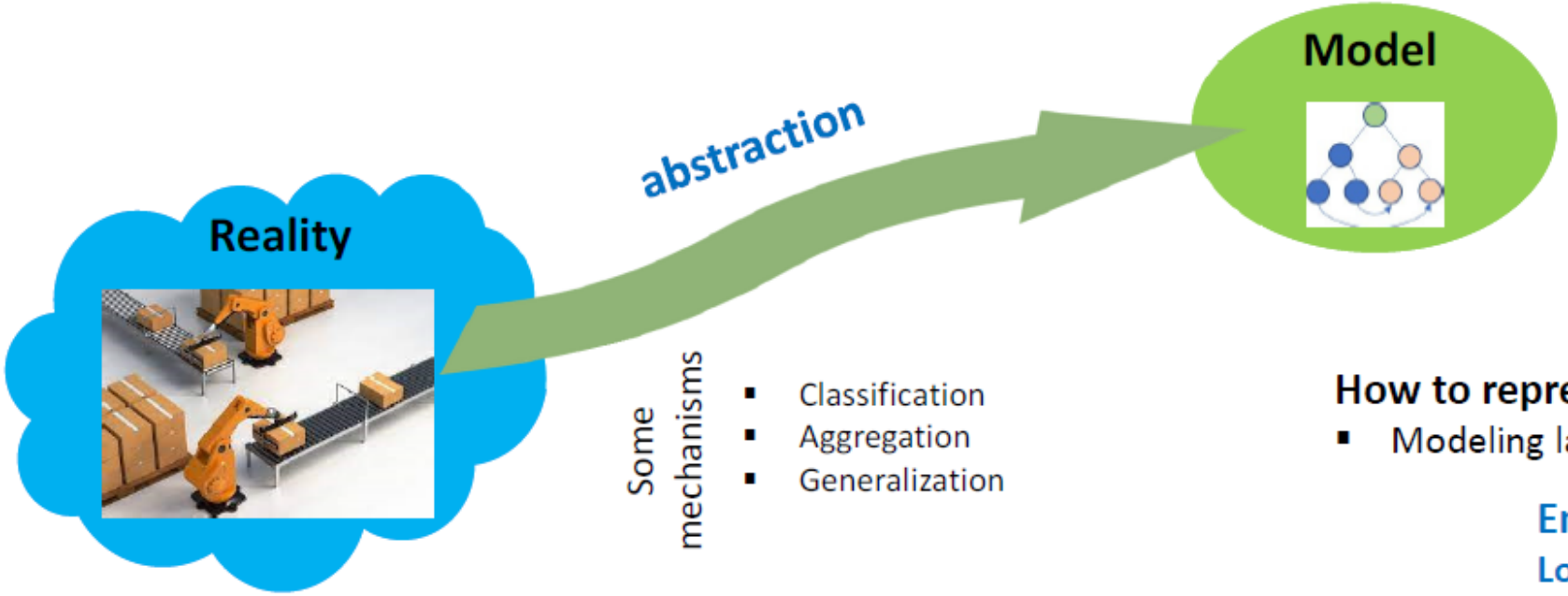
CONTENTS

- **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

© Ana Inês Oliveira, 2024

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?

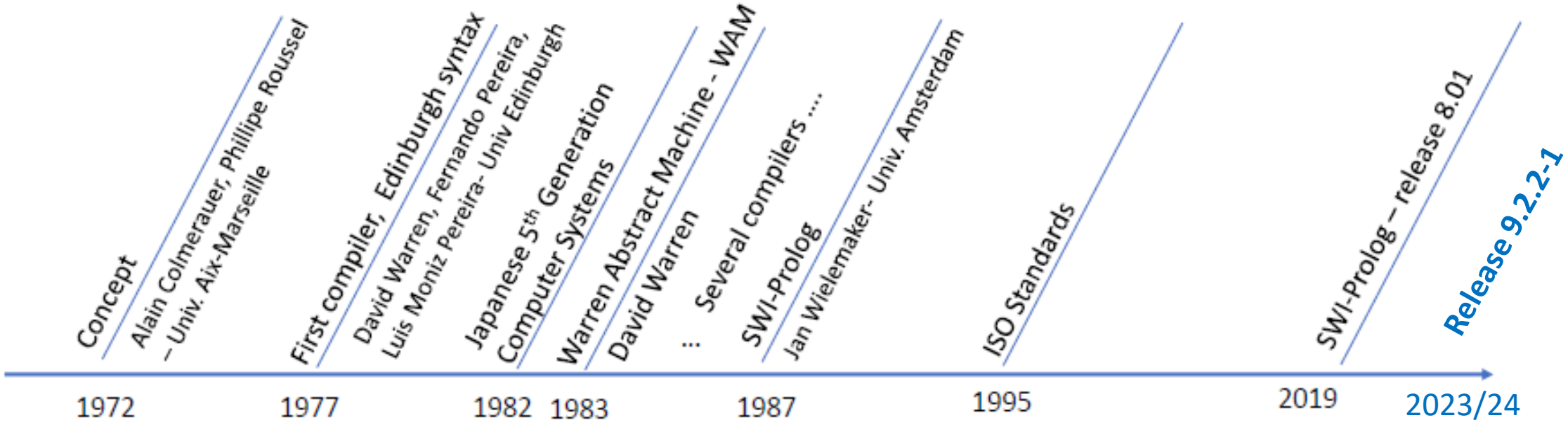
© L.M. Camarinha-Matos 2021-2023

Introduction to the PROLOG Language – History

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

PROLOG

PROgramming in LOGic



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.

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

© Ana Inês Oliveira, 2024

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.

(…)

© Ana Inês Oliveira, 2024

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

© Ana Inês Oliveira, 2024

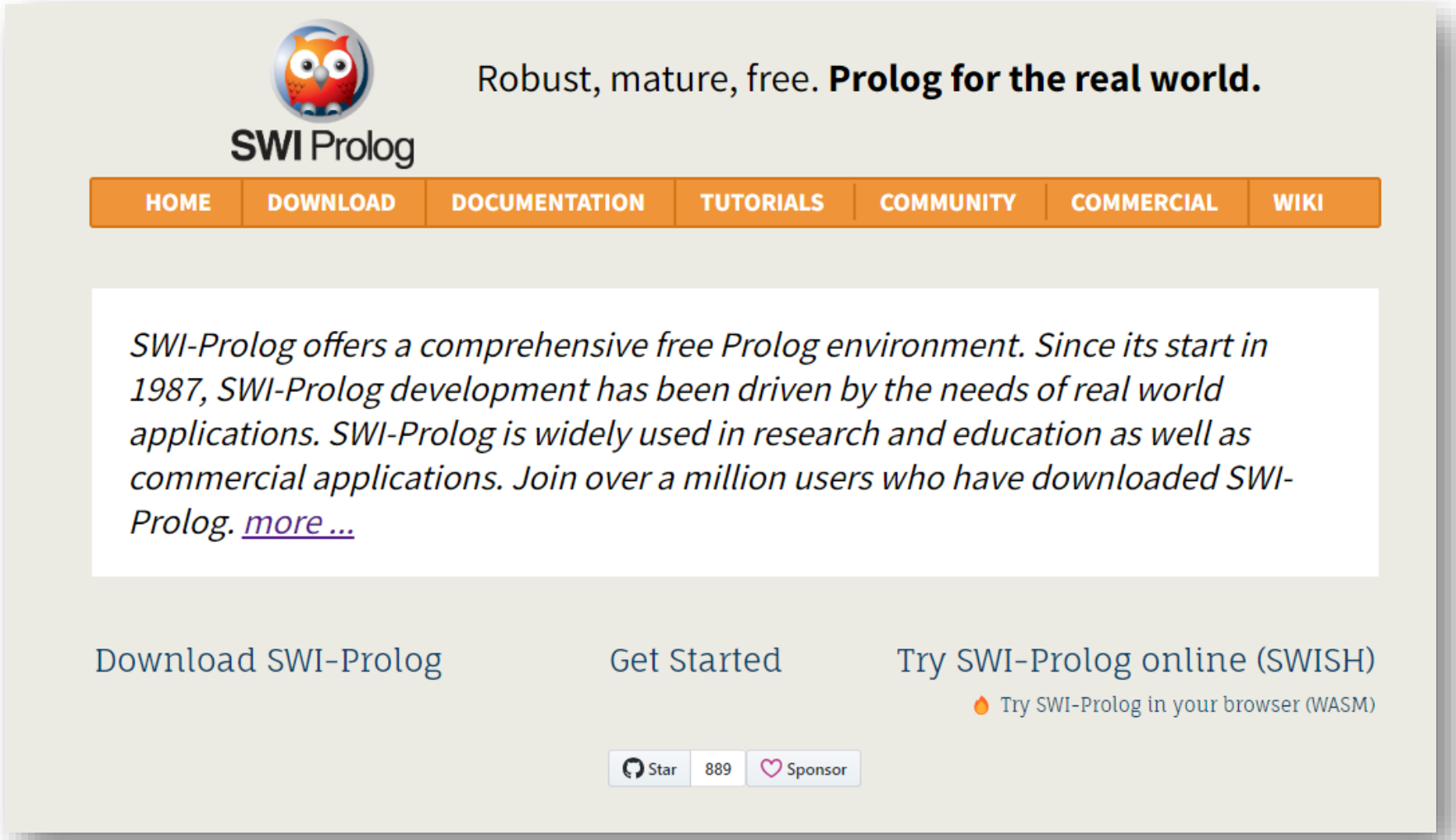
SWI PROLOG

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/



© Ana Inês Oliveira, 2024

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.

https://builtin.com/software-engineering-perspectives/prolog

Facts:

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

Example 1:

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

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

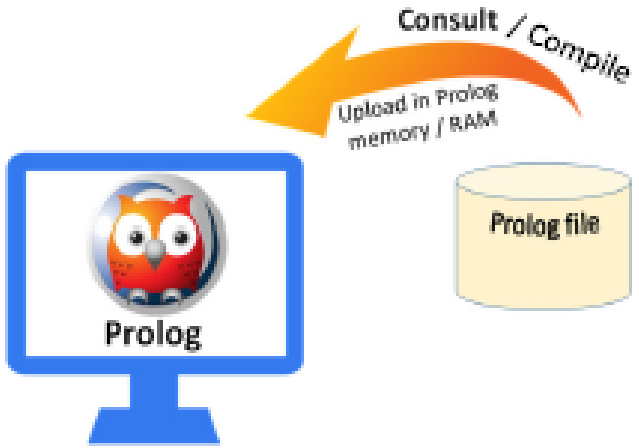
Anything not explicitly stated is considered **false**.

© Ana Inês Oliveira, 2024

PROLOG – Representation of Facts

How to use

1



2 Ask questions → **Queries**

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).

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

constant

variable

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

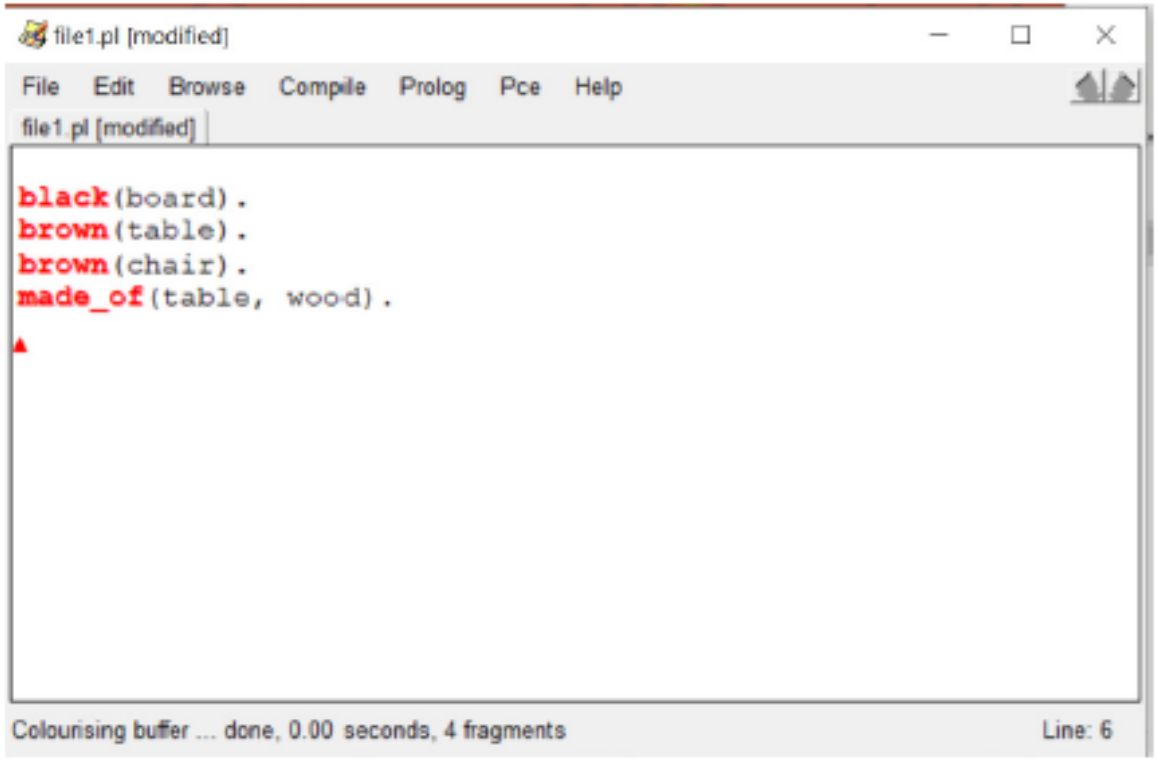
© Ana Inês Oliveira, 2024

PROLOG – Representation of Facts

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

Using SWI-PROLOG

Text editor window:



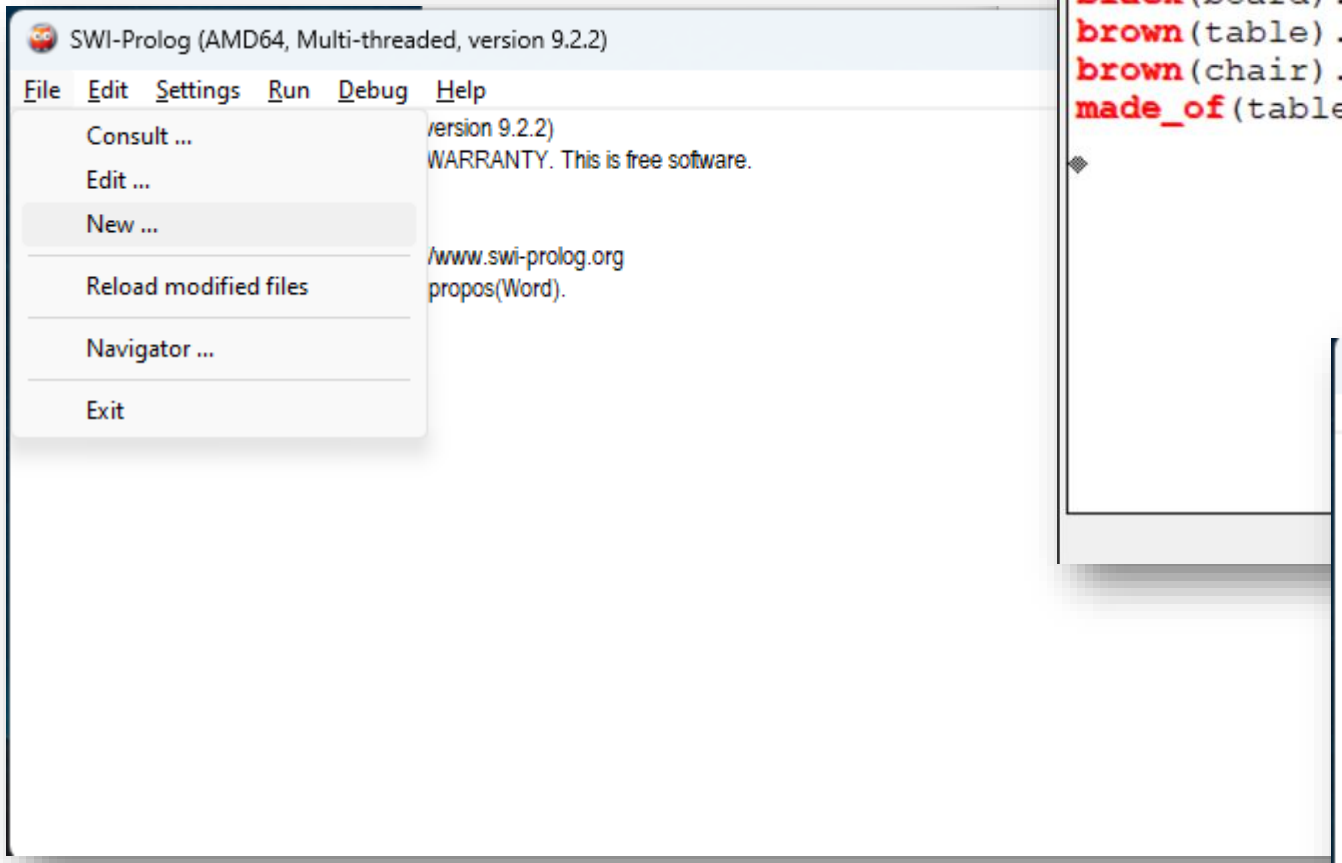
Compile buffer
(or File -> Consult)

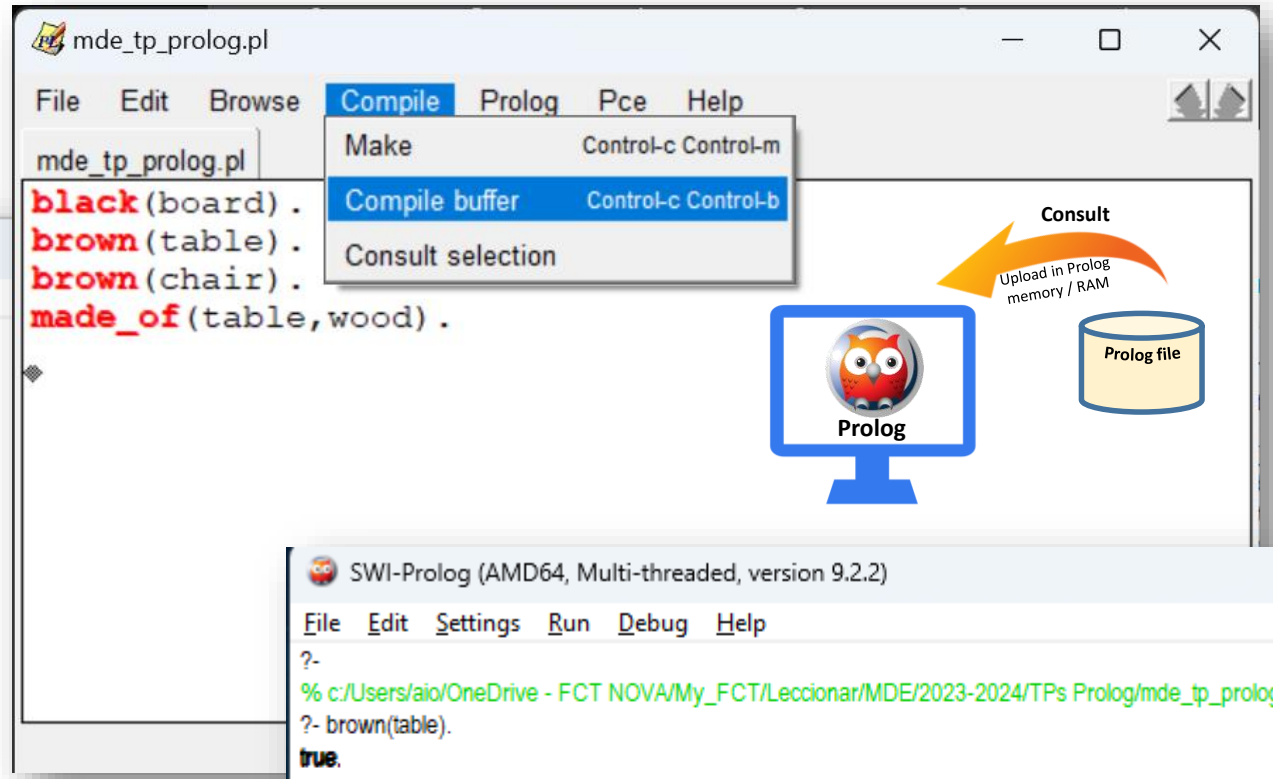
Prolog query window:

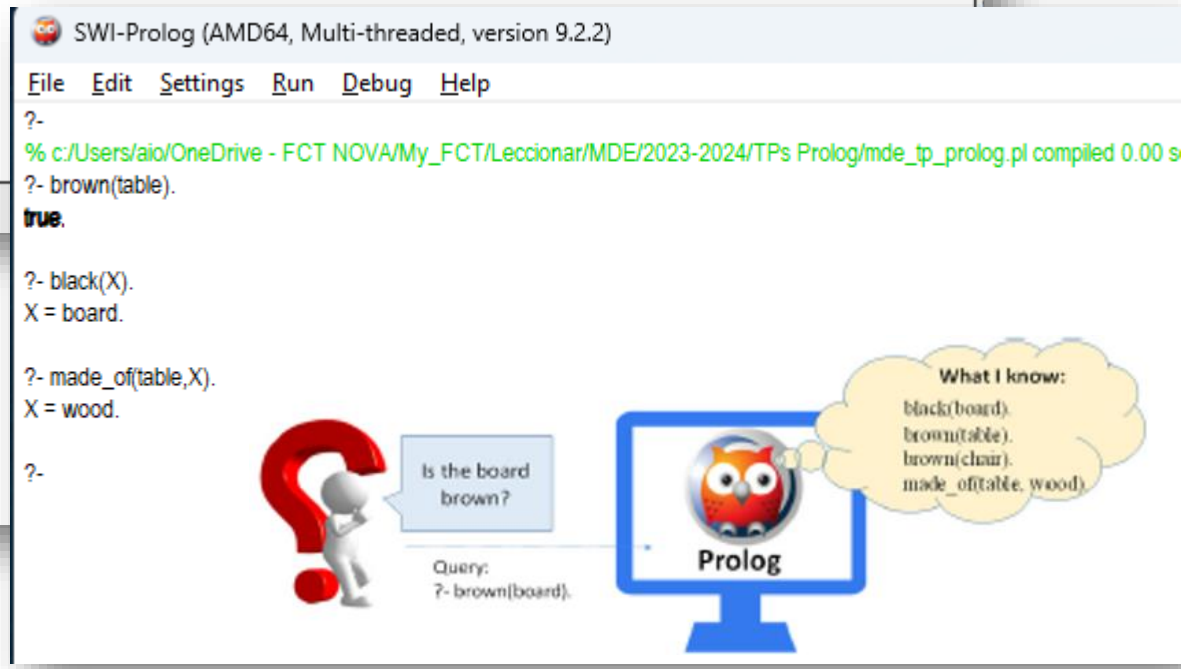
© Ana Inês Oliveira, 2024

PROLOG – Representation of Facts

Using SWI-PROLOG – Lets try!







© Ana Inês Oliveira, 2024

PROLOG - UNIFICATION

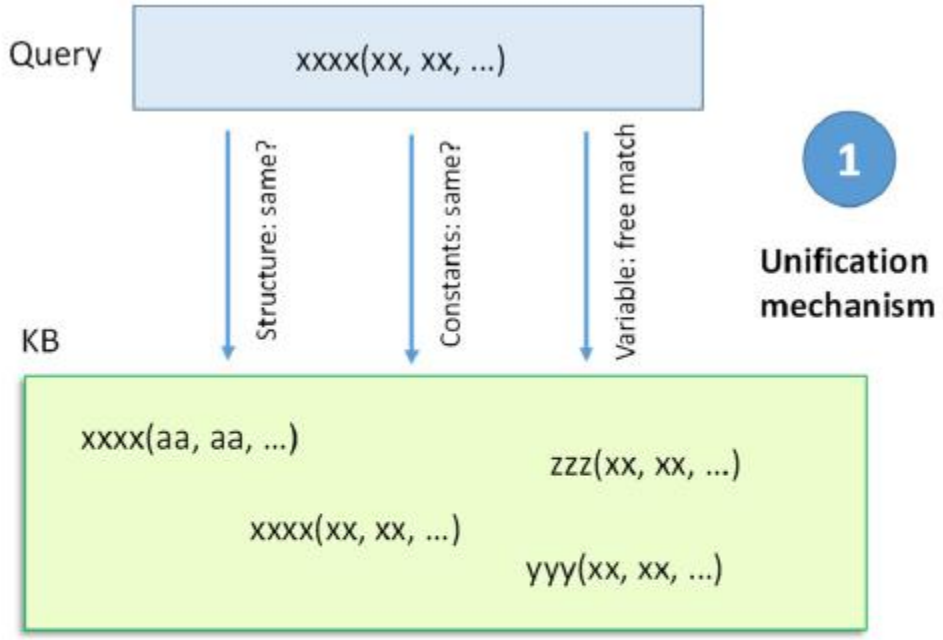
Unification in Prolog is a fundamental concept that involves matching and aligning terms in logic programming

```
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

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

© Ana Inês Oliveira, 2024

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

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

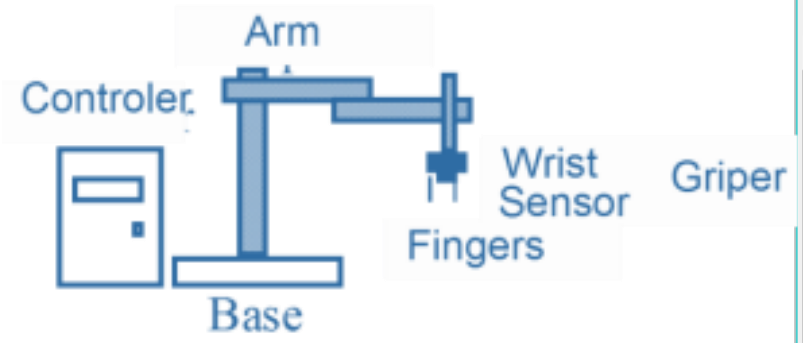
A query can only match a fact when the expression has the same number of parameters (even if anonymous)

© L.M. Camarinha-Matos 2021-2023

© Ana Inês Oliveira, 2024

PROLOG – Another example

Example 1: Model the structure of a robot



example1.pl

File Edit Browse Compile Prolog Pce Help

example1.pl

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

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

Line: 8

Some queries

```
?- 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.
```

Why?

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

SWI Prolog

© Ana Inês Oliveira, 2024

PROLOG – Representation of Rules

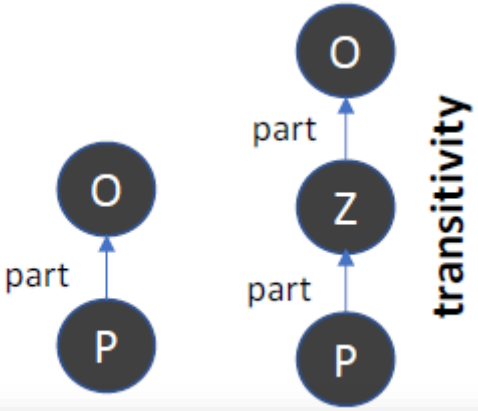
Rules:

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*/



mde_tp_prolog.pl [modified]

```
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

© Ana Inês Oliveira, 2024

PROLOG – Representation of Queries

SWI-Prolog (AMD64, Multi-threaded, version 9.2.2)

File Edit Settings Run Debug Help

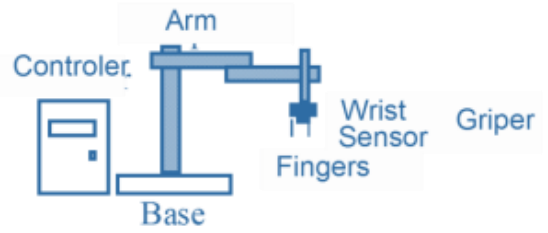
```
?- part(robot,base).
true.

?- 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 .
```

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

© Ana Inês Oliveira, 2024

PROLOG – Representation of Queries

SWI-Prolog (AMD64, Multi-threaded, version 9.2.2)

File Edit Settings Run Debug Help

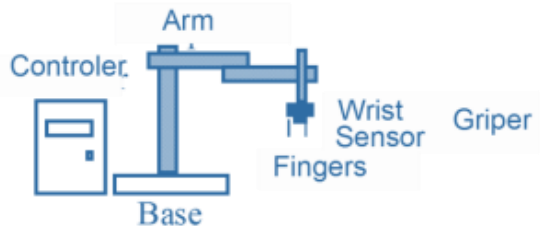
```
?- part(robot,base).
true.

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

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

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

Multiple results




```
?- 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 .
```

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

© Ana Inês Oliveira, 2024

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
R2

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

© L.M. Camarinha-Matos 2021-2023

PROLOG – Representation of Queries

We can generalize the rule


```
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 terms of itself
… i.e., recursive definition

recursion mechanism

© L.M. Camarinha-Matos 2021-2023

PROLOG – Another example

Example 2: Robot components

mde_tp_prolog.pl

File Edit Browse Compile Prolog Pce Help

mde_tp_prolog.pl

```
/* 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).
%...

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

Facts

Rule

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.
```

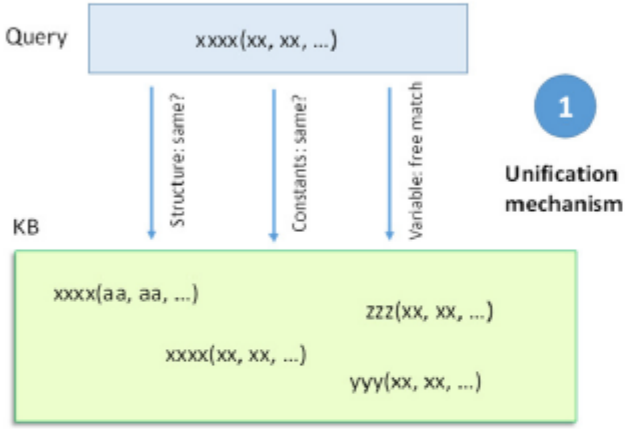
get suitable griper

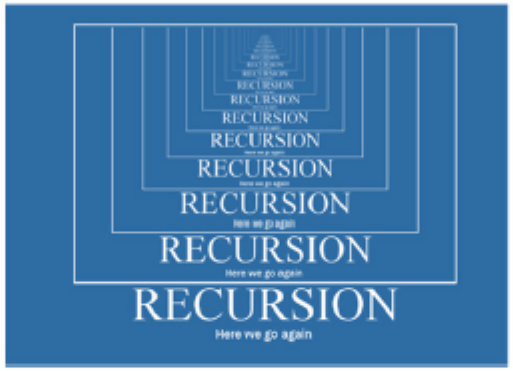
Anonymous variable (represented by underscore) … meaning that we are not interested in it (for this rule)

© Ana Inês Oliveira, 2024

PROLOG – 3 Base Mechanisms

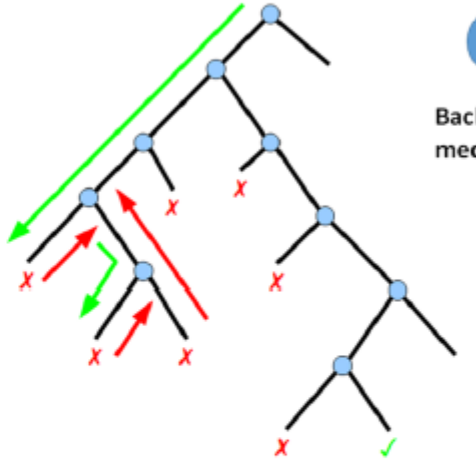
In summary:





2

Recursivity mechanism



3

Backtracking mechanism

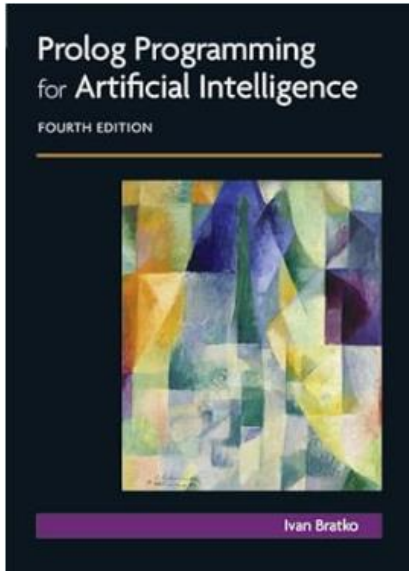
Unification

Recursion

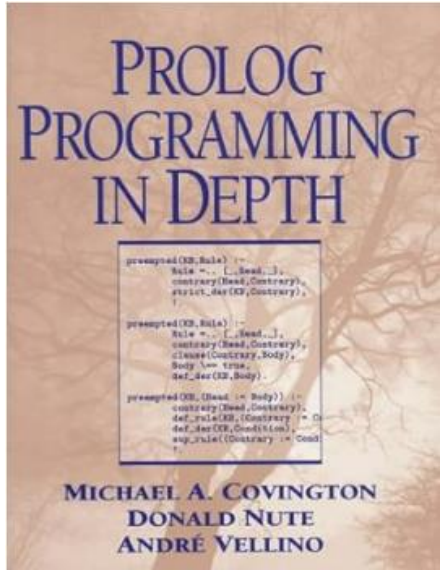
Backtracking

© L.M. Camarinha-Matos 2021-2023

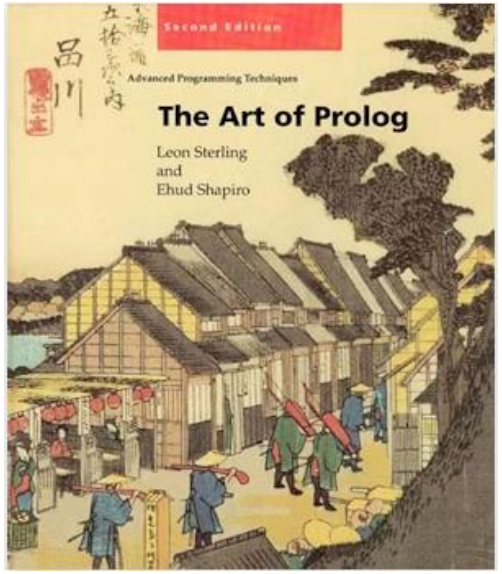
Further reading



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

© Ana Inês Oliveira, 2024